Practical Measurements of Wi-Fi Direct in Content Sharing, Social and Gaming Android Applications

by

Daniël Schoonwinkel

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Electric and Electronic Engineering in the Faculty of Engineering at Stellenbosch University

Department of Electric and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Prof G-J van Rooyen

March 2016

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Signature: .....
D Schoonwinkel

Date: 2015/09/28 .....

Abstract

Practical Measurements of Wi-Fi Direct in Content Sharing, Social and Gaming Android Applications

D Schoonwinkel

Department of Electric and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MEng (E&E)

December 2015

Wi-Fi Direct is a recent expansion to the very successful 802.11 Wi-Fi technology. According to Wi-Fi Alliance, Wi-Fi Direct is being broadly adopted, among others by Google's Android (since version 4.0 Ice Cream Sandwich). However, to the authors' knowledge no formal testing of Wi-Fi Direct throughput, latency, packet loss and energy use on smartphones has been performed.

This thesis presents practical measurements of Wi-Fi Direct capabilities on Android smartphones in practical use-case driven applications. It was found that Wi-Fi Direct would be well suited to content sharing applications and possibly also gaming applications. Furthermore, tests showed that Wi-Fi Direct is more sensitive to communication range and uses more energy compared to standard Wi-Fi. However, with some improvements to this technology, also discussed in this thesis, and on-going development in Android, Wi-Fi Direct could become a reliable and ubiquitous device-to-device communication medium.

Uittreksel

D Schoonwinkel

Departement Elektriese en Elektroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MIng (E&E)

Desember 2015

Wi-Fi Direct is ‘n nuwe uitbreiding tot die 802.11 Wi-Fi tegnologie. Volgens Wi-Fi Alliance, word Wi-Fi Direct al hoe meer ingesluit in elektroniese toestelle, veral in nuwer Android selfone. Android word ontwikkel deur Google, en sluit Wi-Fi Direct funksionaliteit in vanaf weergawe 4.0 Ice Cream Sandwich. Egter, Android se Wi-Fi Direct datatempo, pakkie vertraging, pakkie verlies en energie verbruik is nog nie in ‘n formele akademiese werk gemeet nie.

In hierdie tesis word die vermoeëns van Wi-Fi Direct getoets in praktiese Android toepassings. Die toetse het getoon dat Wi-Fi Direct gepas is vir foto- en videoverspreiding toepassings. Speletjie toepassings kan moontlik ook ondersteun word deur Wi-Fi Direct. Daar is gevind dat Wi-Fi Direct meer sensitief is vir kommunikasie afstand en meer energie intensief is as standaard Wi-Fi. Egter, met sekere verbeteringe (ook bespreek in hierdie tesis) en voordurende ontwikkeling deur Android, kan Wi-Fi Direct as ‘n betroubare en alomteenwoordige ewe-knie kommunikasie medium uitblink.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- My lecturer, Prof Gert-Jan van Rooyen for humour, insight and guidance.
- The Medialab and all my friends and colleagues.
- Manrich van Greunen, my best friend, for working, thinking, and sweating together.
- My family, for supporting me through the easy and difficult times.

Dedications

Aan my Hemelse Vader, wat vir my krag en inspirasie gee elke dag.

Terms of Reference

This project was commissioned by the MIH Media Lab and DSTV / Multichoice, and the following specific objectives were placed on the project:

- Investigate the capabilities of Wi-Fi Direct, possibly as an alternative physical layer that could support content distribution.
- Design and implement a framework that supports distribution of promotional content in places of confluence for example shopping malls, classrooms or conferences.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ADT	Android Developer Tools
AP	Access Point
BER	Bit Error Rate
ICASA	Independent Communication Authority of South Africa
IP	Internet Protocol
IDE	Integrated Development Environment
ISM	Industrial, scientific and medical radio band
OS	Operating System
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
P2P GO	Peer-to-Peer Group Owner
PIN	Personal Identification Number
QR	Quick Response Code
SSID	Service Set Identifier
SNR	Signal-to-Noise Ratio
TCP	Transmission Control Protocol
SoftAP	Software Access Point
UI	User Interface

Units

kbps	Kilobits per second
kB	Kilobytes
Mbps	Megabits per second
MB	Megabytes
ms	Milliseconds
m	Meters
dB	Decibels

Variables

M_s	Number of successfully received bytes of a message
T_r	The time at which the reply packet arrives
T_s	The time at which the packet was sent
P_r	The number of successfully received packets
P_s	The total number of sent packets
$\bar{x}$	Sample mean
s	Sample standard deviation
s_m	Estimate of standard error
N	Sample size
$t_{95\%}$	95% Student's t distribution value for a given degree of freedom.
ME	Margin of Error (width of confidence interval)

Font types

WifiP2PApp	This font type refers to a specific module or functional block.
WifiP2PApp	This font type refers to an app as a whole.
`WifiP2PApp`	This font type refers to a specific Java class.

Chapter 1

Introduction

Wi-Fi Direct is an adaption of standard 2.4 GHz Wi-Fi, enabling direct device-to-device communication on various devices including Android smartphones. Device-to-device communications present a unique opportunity for users to share services without the need for pre-existing infrastructure. The purpose of this thesis is to test Wi-Fi Direct, by implementing a Wi-Fi Direct framework and a set of use-case applications, and testing throughput, latency, packet loss and energy use while using the applications. This will provide Android application developers insights into Wi-Fi Direct's performance and capabilities.

1.1 Background

Entertainment content is available through various media channels including live television, DVDs and Blu-ray, computer and smartphone internet, and live radio. According to Nielson [58], live TV is still the dominant channel of consumption, but from 2013 to 2015 computer and especially smartphone viewing have increased, and live TV use has decreased. This indicates that smartphones would be a useful platform to explore as a growing content distribution channel.

The number of smartphones are estimated to be over 1 billion devices worldwide [13]. Smartphones are becoming more powerful [60] and are already capable of executing various tasks, for example email, internet browsing, music and video playback and games [60]. South African smartphone penetration is estimated to be above 30%, and projected to be 47% by 2017 [29].

According to another study by Nielson [27], the top categories for smartphone applications (applications) use is social media, entertainment and communication. In these areas, entertainment has seen the greatest increase (71%) between 2012 and 2013.

In the smartphone market, Android has a very large influence. By providing an open-source operating system available on various hardware platforms including Samsung, HTC, LG and others [44], Google has revolutionised the

smartphone market and gained a significant userbase. According to Gartner [12], 2014 international market share for Android was 80.7% compared to iOS at 15.4% and Windows Phone at 2.8%. MyBroadband [2] performed a similar study and found Android holding 57.8% market share in South Africa.

Primarily, data connections which provide social and entertainment services use GPRS EDGE and 3G/HSDPA, with 4G becoming more widely used [25]. Although these technologies offer high throughput, problems like high latency and poor consistency as well as high data costs still exist [46]. However, most smartphone devices are also equipped with Wi-Fi connectivity, which enables the smartphone to connect to a higher-throughput Wi-Fi access point when it is available in the local area.

Similar to Wi-Fi, Wi-Fi Direct [39] is a 2.4-GHz communications protocol based on the IEEE 802.11 standard [47]. However, Wi-Fi Direct functions without a fixed access point and each device acts as either a Wi-Fi software access point (SoftAP) or as a Wi-Fi client, in this way creating a peer-to-peer network. Furthermore, network services can be advertised on link-layer (explained further in Chapter 2), enabling tasks such as wireless printing and screen sharing (Miracast [61]) to be performed without extensive setup. Google implemented Wi-Fi Direct functionality in Android 4.0 [1], enabling the use of Wi-Fi Direct in applications.

It is the purpose of this thesis to investigate the potential of Wi-Fi Direct as a content distribution medium between Android smartphone devices, as well as Wi-Fi Direct's suitability to other applications such as gaming and social applications. This purpose is more formally outlined in the next section.

1.2 Problem Statement

Wi-Fi Direct is being supported by an increasing number of Android devices according to Android Play Store statistics [8]. However, to the authors' knowledge, no practical studies have been conducted which measure Wi-Fi Direct throughput, latency, packet loss and energy use in Android applications. In this thesis, we perform such practical measurements to provide useful insights to Android application developers considering Wi-Fi Direct as a device-to-device communications medium in content sharing, gaming and social applications.

Furthermore, due to the mobility of smartphones and the frequent changing of communication opportunities (as smartphones users move into range and out of range of each other), the Android applications need to compensate for this. In this thesis, we present a framework which supports fast and simple discovery and connection management.

1.3 Related Work

Related work to this thesis includes work by Camps-Mur et al. [45] which provide experimental results of Wi-Fi Direct timing, energy use and throughput on Wi-Fi Direct enabled laptops as well as work by Trifunovic et al. [59] which compare the energy efficiency of a three 2.4 GHz communication protocols on smartphones. Two related projects to this thesis, PeerDeviceNet [23] and AllJoyn [18] are also discussed here, as both provide Wi-Fi Direct frameworks which could potentially be used to measure Wi-Fi Direct performance.

1.3.1 Camps-Mur et al. Overview and Experimentation of Wi-Fi Direct

Camps-Mur et al. [45] present an in-depth overview of the Wi-Fi Direct protocol. Using two laptops as Wi-Fi Direct nodes, they measure Wi-Fi Direct connection time. They also measure Wi-Fi Direct energy use and throughput during a test where one laptop is acting as a tethered 3G connection to the Internet, i.e. forwarding Internet traffic through Wi-Fi Direct.

Camps-Mur et al. presents connection time and throughput measurements comparable to values found in this thesis, however we perform tests on Android smartphone devices, instead of laptops.

1.3.2 Trifunovic et al. Fair and Efficient Energy Usage in Device-to-Device Communication

Trifunovic et al. [59] present an experimental study showing the different power usages of three 2.4 GHz communication protocols, namely Bluetooth, Wi-Fi Direct and WLAN-Opp (standard Wi-Fi, where one device is set-up as a Wi-Fi hotspot). The protocols are compared and results show that Bluetooth far outperforms Wi-Fi Direct and WLAN-Opp in terms of energy efficiency, with WLAN-Opp slightly more energy efficient than Wi-Fi Direct. These energy-use experiments provide insight for application developers considering which communications protocol to use, but lacks the throughput-to-energy trade-off between Bluetooth, Wi-Fi Direct and WLAN-Opp. This thesis will include tests measuring throughput and energy use of Wi-Fi Direct and Bluetooth to give such trade-off insights to application developers.

1.3.3 PeerDeviceNet

PeerDeviceNet [23] was created to share content securely between devices of family, friends or colleagues. Connections are secured using Secure Socket Layer encryption and as devices are connected directly, they are secured against a man-in-the-middle attack or similar threats. The PeerDeviceNet framework

selects one device as the Wi-Fi Direct SoftAP and generates a Quick Response (QR) code containing the Wi-Fi Direct Service Set Identifier (SSID) and password. Other devices can then connect to the Wi-Fi Direct SoftAP by scanning the QR code to obtain the connection credentials. This ensures that a secure connection is established, but requires close proximity to the Wi-Fi Direct SoftAP device. Although PeerDeviceNet provides better security than the framework presented in this thesis, our framework is designed to create Wi-Fi Direct connections automatically to better utilise available connection opportunities. This design choice is explained further in Chapter 3.

1.3.4 AllJoyn

AllJoyn is an open-source framework which can use various communication media such as Wi-Fi, Ethernet, serial and power-line communications. AllJoyn is designed to be agnostic to communication media, device operating system (it supports RTOS, Arduino, Linux, Android, iOS, Windows, and Mac) and programming languages (C, C++, Obj-C, and Java). This framework could be used to perform measurements of Wi-Fi Direct capabilities, but it was decided against, as the framework abstraction obscures the lower-level Wi-Fi Direct performance. The framework proposed in this thesis only supports Wi-Fi Direct, measuring the capabilities of Wi-Fi Direct without the complexities and added CPU and memory use of a large framework implementation like AllJoyn.

1.4 Objectives

The goal of this project is to evaluate Wi-Fi Direct as a possible device-to-device communications medium for Android applications, specifically content sharing, games and chatting. This was done in the following steps:

- Design and implement a Wi-Fi Direct framework which Android application developers can use to facilitate communications, which supports dynamic discovery and connection management.
- Adapt and implement three sample applications: a file transfer application for content sharing, a game application and a chat application, to use the Wi-Fi Direct framework.
- Measure throughput using the file transfer application, latency and packet loss using the game and chat applications.
- Compare Wi-Fi Direct and standard Wi-Fi by running the same tests through standard Wi-Fi.

- Measure energy use while connected via Wi-Fi Direct and during long running applications such as file transfers.

As a secondary objective of this thesis, recommendations and possible difficulties of Wi-Fi Direct on Android will be discussed.

1.5 Summary

In this chapter, we discussed the environment of Android smartphones and how Wi-Fi Direct could be a medium for enabling content sharing and other services. We discussed the related work such as studies by Camps-Mur et al. and Trifunovic et al, as well as PeerDeviceNet and AllJoyn frameworks implementations similar to our framework.

This thesis is focussed on measuring the capabilities of Wi-Fi Direct, by implementing a framework for content sharing, games and social applications, and testing Wi-Fi Direct's suitability to each of these applications.

In the next chapter we will discuss the concepts from literature needed to better understand the implementation of the framework. Chapter 3 describes the higher level system design and Chapter 4 describes the actual implementation details (as it was done in Android). Chapter 5 discusses the adaption and implementation of applications used in the testing of Android Wi-Fi Direct. Chapter 6 covers all of the tests completed on the framework. We conclude in Chapter 7, discussing the results obtained in Chapter 6 in terms of our objectives obtained, as well as pointing out some Android Wi-Fi Direct pitfalls found during framework implementation. Finally, we comment on the future of Wi-Fi Direct in general.

Chapter 2

Literature Study

In this chapter we discuss device-to-device networking technologies commonly found in smartphones, three common network protocols (UDP, TCP and FTP), network measurement metrics and the Android and Apple iOS operating systems. These concepts are used when describing the implementation of the Wi-Fi Direct framework in the next chapter.

2.1 Network Technologies

In this section, we discuss device-to-device networking technologies such as Wi-Fi Direct and Bluetooth.

2.1.1 Wi-Fi Direct

Wi-Fi Direct [63] is a Wi-Fi Alliance [39] standard that enables IEEE 802.11 [16] wireless communication between supporting devices, without the need for an access point. Each device forming part of the Wi-Fi Direct network (called a group) can act either as a Peer-to-Peer Group Owner (P2P GO) or P2P Client. The P2P GO is responsible for announcing the group through beacons and assigning IPs to the connected P2P Clients, fulfilling the equivalent role of a Access Point (AP) in 802.11 infrastructure mode. P2P Clients connect to the P2P GO similarly to a standard Wi-Fi client connecting to an AP. A device can at the same time be connected to a Wi-Fi AP and a Wi-Fi Direct group, but not to more than one Wi-Fi Direct group at a time.

The roles of P2P GO and P2P Client are resolved at the time of group formation, depending on each device's P2P GO Intent. The device with the highest P2P GO intent becomes the P2P GO. The role of P2P GO is more power intensive [59] and is therefore usually handled by a device with higher energy capacity, for example a laptop instead of a smartphone.

Wi-Fi Direct is secured with Wi-Fi Protected Setup (WPS) [62] using a PIN or push-button configuration (user accepts incoming connections).

Wi-Fi Direct, in accordance to the IEEE 802.11 standard, operates in the 2.4 GHz Industry, Scientific and Medical (ISM) frequency band [52]. The ISM band is a an unlicensed frequency band, but devices operating in this band are limited to 100 mW output power [31]. Wi-Fi Direct can support the 802.11a/b/g/n standards, which can deliver a theoretical maximum speed of 250 megabits/second (Mbps) using the 802.11n standard. The range at which Wi-Fi Direct devices can connect is estimated at 200 m [39].

Wi-Fi Direct Service Discovery

A unique feature of Wi-Fi Direct, in contrast to standard Wi-Fi, is the ability to perform service discovery before the devices are connected to the same P2P Group. This is possible due to the Generic Advertisement Protocol (specified in IEEE 802.11u [15]) running in the link layer. This protocol allows application layer implementations to advertise services, which applications running on other devices might want use, for example a file sharing service. Because connecting to a Wi-Fi Direct GO requires a significant amount of energy [59], scanning for relevant services before connecting can avoid the unnecessary energy expenditure of P2P group formation when services are not required by the user.

According to Wi-Fi Alliance, standard Wi-Fi Direct services include sending and receiving content between Wi-Fi Direct devices, Miracast [61] screen sharing and printing to Wi-Fi Direct capable devices.

2.1.2 Bluetooth

Bluetooth is an ad-hoc transmission standard that was defined by Ericsson [10] in 1994. According to Haartsen [51], it was designed to support short-range communication between devices such as PCs, laptops, cellphones and other peripherals. As of 2014, 24000 companies are part of the Bluetooth Special Interest Group, the governing body of the Bluetooth standard [20].

Bluetooth networks, called piconets, are formed by a master device broadcasting its identity packet on various frequency bands in the 2.4 GHz ISM band. The Bluetooth protocol uses random frequency hopping to efficiently use the available bandwidth and avoid interference. This is why the slave devices need to detect the identity packets during their scan cycles and use them to synchronise their frequency hopping to the master device's pattern. Once the devices agree on the random frequency hopping scheme, the master and slave can communicate. The master is responsible for administrating the connection by giving each slave device its allotted transmitting time slot [51]. Bluetooth V1.0 is limited to theoretical data rates of 1 Mbps, but with improvements to the protocol, a theoretical data rate of 24 Mbps is possible by communicating in a co-located 802.11 channel. However, this capability is only

available for devices satisfying the Bluetooth V3.0 + HS standard [50]. The maximum range of Bluetooth connections are estimated at 100 m [10].

2.1.3 Network Technologies Summary

	Max Range	Bitrate
Wi-Fi (Direct)	250 m	1 - 250 Mbit/s
Bluetooth	100 m	1 - 24 Mbit/s

Table 2.1: Comparison of popular communications technologies. Values given in this table are theoretical estimates and do not necessarily represent user experience values.

2.2 Networking Protocols

In this section we discuss two common networking protocols based on the Internet Protocol [53]. Although many other protocols exist, these two protocols represent two diverse paradigms of network communication. We also discuss the File Transfer Protocol, used later in the content sharing application.

2.2.1 User Datagram Protocol

The User Datagram Protocol (UDP) is defined in RFC 768 [54] as a minimal transport mechanism. UDP is described as transaction orientated, i.e. messages are sent in simplex, best-effort fashion without delivery guarantee and duplicate protection. UDP uses Internet Protocol (IP) [53] addressing to route messages and port numbers to facilitate multiple applications communicating through a single network interface. Because there is no acknowledge mechanic in UDP, UDP packet sizes are specified in the packet header. This causes UDP payloads to be limited to 64 kBs as the packet size field is a 16-bit number.

Java provides an interface for transmitting over UDP sockets with the `DatagramSocket` [9] class.

2.2.2 Transfer Control Protocol

In contrast to UDP, the Transfer Control Protocol (TCP) is a connection-orientated transmission protocol, i.e. a full-duplex connection providing end-to-end acknowledge, error correction, duplicate detection, sequencing and flow control. TCP (as defined in RFC 675 [48]) is capable of sending payloads in multiple packets and reassemble the payload at the receiving end. This enables TCP to send large payloads reliably, in contrast to UDP. Java implements TCP communications with the `ServerSocket` [26] and `Socket` [28] classes.

2.2.3 File Transfer Protocol

The File Transfer Protocol (FTP), defined in RFC 765 [55], is an application level protocol used for transferring files between networked devices. FTP uses two channels, namely the command channel and data channel. The command channel (using text commands) is used to set-up and control the flow of information in the data channel. The data channel can be used to send text or binary information. FTP can run on UDP and TCP, but according to the updated FTP specification in RFC 959 [56], FTP assumes the underlying protocol is TCP, which ensures the reliable transmission of files.

2.3 Network Metrics

In order to compare communication channels, three metrics namely throughput, latency and packet loss, are defined as follows:

Throughput:

Throughput is the rate of successful data delivery over a channel. Throughput is calculated by the following equation:

$$\text{Throughput} = \frac{M_s}{T} \tag{2.3.1}$$

where M_s is the number of successfully received bytes of the message, and T is the time taken to receive the message. Throughput is usually measured in kilobits/second [34].

Latency:

Latency is defined as the round-trip time between when a request is sent and an answer to that request is received. Latency is calculated by the following equation:

$$\text{Latency} = T_r - T_s \tag{2.3.2}$$

where T_r is the time that the reply packet arrives, and T_s is the time that the original packet was sent [41].

Jitter:

Jitter (also known as packet delay variation) is defined as the deviation in latency of a packet stream. Jitter is calculated by the following equation:

$$\text{Jitter} = L_2 - L_1 \tag{2.3.3}$$

where L_2 is the latency of the second packet, and L_1 is the latency of the first. [17].

Packet loss:

Packet loss is defined as the number of packets that did not successfully reach the intended destination, sometimes also expressed as packet loss ratio:

$$\text{Packet loss} = P_s - P_r \tag{2.3.4}$$

and

$$\text{Packet loss ratio} = 1 - \frac{P_r}{P_s} \tag{2.3.5}$$

with P_r the successfully received packets and P_s the total number of sent packets [22].

2.4 Android Operating System

Android is an open-source Operating System (OS) capable of running on mobile phones, tablets, wearable devices, and in-car and home entertainment systems. Android OS is based on the open-source Linux kernel and is actively being developed by Google. Android has a wide user base: 1.5 billion applications and games are downloaded from the Google Play store per month [3].

Android Developer Tools (ADT) provides a Java IDE for creating applications and code generation from XML to specify UI layout. The Android libraries provides a hardware abstraction layer and libraries needed so that Android applications can be run by a wide range of devices. The custom Android Dalvik Virtual Machine (Dalvik VM), runs the Java code. The Dalvik VM is specifically designed for embedded environments, optimised for efficient CPU and memory use [43].

2.4.1 Android Design Patterns

As in most high level programming languages and environments, there are design patterns and methodologies that need to be understood before code can be written and the required libraries can be used. Three of the Android design patterns (`Activity`, `Service` and `BroadcastReceiver`) used in this project will be explained here.

2.4.1.1 `Activity`

The Android `Activity` class represents a front-end process that the user will interact with. The `Activity` is started when a user selects it from the applications menu and gets suspended when the user returns to the home screen. `Activitys` can be killed while they are suspended by the Android OS to free memory, and are thus only guaranteed to be alive while the user is interacting with them. This makes them ideal for short bursts of interaction, but unreliable for long running processes.

2.4.1.2 `Service`

In contrast to the `Activity` class, the Android `Service` class represents a back-end process for handling long-running tasks. Depending on the requirements of the `Activity`, Android `Service`s can either be started by or bound to an `Activity`.

Started Services run until they are stopped by an `Activity` or the `Service` stops itself. An example of a started service would be a music player which is started by the music player `Activity`, but still plays music in the background after the `Activity` has been closed.

Bound Services run as long as an `Activity` is bound to it. An example of a bound service is a service which downloads content for the `Activity` in the background, updating the user interface (UI) while the `Activity` is active, but stopping when the user no longer requires the updates.

2.4.1.3 Android Interface Description Language

The Android Interface Description Language (AIDL) can be used to describe an abstract interface to a bound service, limiting the functionality exposed to an `Activity`, and thus decoupling the implementation of the service. AIDL also enables the `Activity` to be compiled without the specific `Service` code, but rather an interface descriptor file. The `Service` functionality is linked at run-time by the Android system.

It is important to note that the `Service` is a class in the Android environment performing long-running tasks, which should not be confused with the P2P services mentioned earlier. A P2P service is a high-level description of a capability that **Provider**s make available to the network, for example video content distribution.

2.4.1.4 `BroadcastReceiver`

A `BroadcastReceiver` is an Android class which enables asynchronous broadcasts to be received from other processes running on the device. These broadcasts can be used to respond to events like connection status changes, service discoveries and incoming messages.

2.5 General Design patterns

In addition to the Android specific design patterns mentioned above, two other general object-orientated design patterns were used. These design patterns are described in the book *Design Patterns* by Gamma et al. [49].

2.5.1 Bridge pattern

The purpose of a Bridge object is to decouple the implementation of two objects, so that each can vary independently. In this project, this pattern is used to decouple the UI code used in `Activity` classes from the code used to communicate to the Wi-Fi Direct framework.

2.5.2 Flyweight pattern

The Flyweight pattern is used when an object needs to be shared between many objects. The Flyweight object contains intrinsic information independent of which object is currently using it. The objects using the Flyweight translate the information into their context.

2.6 Apple iOS

Apple runs iOS on all of its mobile devices, including iPhones and iPads. Unlike Android, iOS is closed source, however the Xcode IDE is available for use free of charge on Mac computers. The iOS App Store has over 100 billion downloads since its launch in July 2008 [5]. Apple announced at the beginning of 2015 that it has sold 1 billion iOS devices [4].

Although this project focusses on testing Wi-Fi Direct capabilities on Android, it is important to note that iOS also supports Wi-Fi Direct. As of iOS 7 (released in September, 2013), applications are able to use Wi-Fi Direct as a communications medium through the Multipeer Connectivity framework [19]. In chapter 3 we will investigate Wi-Fi Direct capabilities of Android and iOS.

2.7 Summary

In this chapter, we discussed networking technologies, network protocols, programming design patterns and the Android and Apple iOS operating systems which will be used in our system design.

In the following chapter we discuss the system design of the framework as was set out in the problem statement and objectives of Chapter 1.

Chapter 3

System Design

In this chapter we discuss the possible use-cases of a Wi-Fi Direct framework, followed by technical specifications. Android and iOS were investigated as possible platforms for implementing the Wi-Fi P2P framework, however the final implementation was done in Android. Throughout the chapter, we give an oversight of the design choices made while developing the framework.

This chapter represents the high-level design of the framework and device interactions while the detail implementation is discussed in the next chapter.

Please note: Android refers to Wi-Fi Direct as Wi-Fi P2P, a convention that we will follow when referring to Wi-Fi Direct running on mobile devices.

3.1 Use-cases

In this section we discuss three example use-cases of a Wi-Fi P2P framework, namely content sharing, gaming and social (chatting). The choice of use-cases are based on the Nielson study of smartphone application usage [27] which suggest that these are common uses for smartphones. Furthermore, these use-cases each highlight a different aspect of Wi-Fi P2P communication capabilities in terms of throughput, latency, jitter and packet loss.

3.1.1 Content Sharing: Figure 3.1

The user interested in receiving entertainment content establishes the Wi-Fi P2P network. Other users in range, able to offer entertainment content, connect to this network and make the content available to the network.

3.1.2 Gaming: Figure 3.2

The user interested in playing a game establishes the Wi-Fi P2P network and hosts the game. One or more (depending on gaming application) connect to the network and join the game.

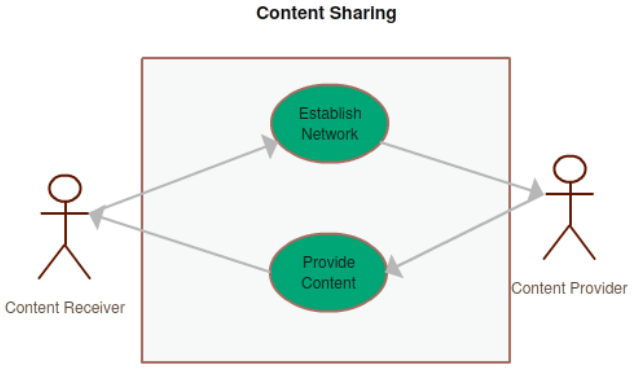

Figure 3.1: Content sharing use case. The content receiver establishes the Wi-Fi P2P network and the content provider shares the entertainment content with the network.

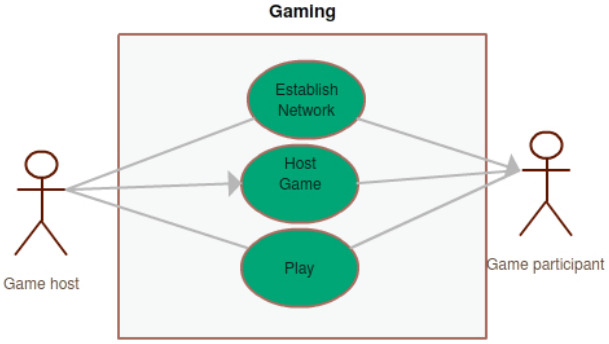

Figure 3.2: Gaming use case. A user interested in playing a game establishes Wi-Fi P2P network and hosts a game application. Other (one or more) users can then connect to this game application and start playing.

3.1.3 Chatting: Figure 3.3

The user interested in starting a chat conversation establishes the Wi-Fi P2P network. Multiple users can join this network and all chat simultaneously to all other users.

3.2 Framework Requirements

Using the above mentioned use-cases and objectives stated in Chapter 1, we define functional requirements for the Wi-Fi P2P framework below.

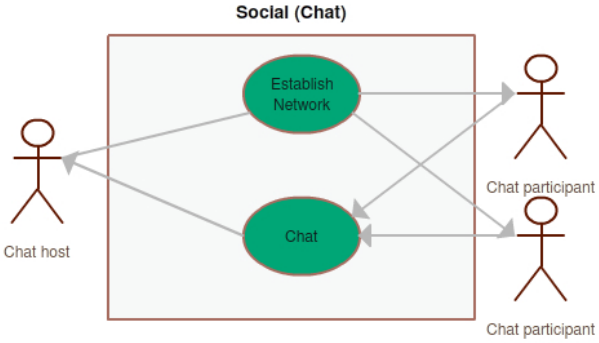

Figure 3.3: Chatting use case. A user interested in chatting establishes Wi-Fi P2P network, to which other users connect. All connected users chat can simultaneously.

3.2.1 High throughput

According to Wi-Fi Alliance, Wi-Fi P2P, can provide for significant throughput (up to 250 Mbps on 802.11n [39], similar to standard Wi-Fi). High throughput would benefit content sharing, as more content can be shared in a shorter timespan. The Wi-Fi P2P framework should therefore leverage this available throughput to provide users with the fastest downloads possible.

3.2.2 Low Latency, Jitter and Packet Loss

The Wi-Fi P2P framework should have low latency, in order to support gaming and video streaming applications. According to [42], 100 ms is an acceptable latency for gaming applications. According to [24], a latency of below 150 ms is acceptable for interactive (streaming) video. Similarly jitter needs to remain low, to give a reliable stream. Jitter below 50 ms is acceptable for gaming [42], 30 ms are acceptable for interactive video [24].

Packet loss should be as low as possible, as it greatly reduces user experience. For gaming, a packet loss of less than 5% is acceptable, which is equivalent to missing one frame in a 20 frames per second game. According to [24], a packet loss of less than 1% is recommended for video streaming applications.

3.2.3 Multiple Device Support

All of the mentioned use-cases require other users in close proximity and hence it is assumed that this framework will be used in areas where there are frequently a gathering of large numbers of people, for example shopping malls, classrooms or conference venues. In such busy areas, the Wi-Fi P2P framework should support connecting multiple devices to use all available connection opportunities.

3.2.4 Dynamic Network Management

As was mentioned above, the framework will most likely operate in a busy and dynamic environment. The framework therefore needs to be able to adapt to changing network situations by frequently discovering new connections opportunities as well as sustaining (keeping alive) current connections.

3.2.5 Modular Design

The framework should be modular, i.e. the framework code and process should be separate from the applications using the framework. This requirement decouples the implementation of applications and framework, so that either can be changed at a later stage.

3.2.6 Reasonable Energy Use

It is known from literature that Wi-Fi P2P consumes more energy than standard Wi-Fi and Bluetooth [59]. When possible, the framework should strive to limit energy usage.

3.3 Mobile Operating System Platform

Before we explain the Wi-Fi P2P framework system design, we need to investigate what functionality the different platforms, Android and iOS, provide for connecting with Wi-Fi P2P.

Although the Android and iOS methods for handling Wi-Fi P2P connections differ, some similarities exist. In the following subsections, we will compare the methodologies of the two operating systems. Similarities between Android and iOS functionality is summarised in Table 3.1.

Table 3.1: Summary of Wi-Fi P2P terms

Android	iOS
Wi-Fi P2P Group (`WifiP2pGroup`)	Multipeer Session (`MCSession`)
P2P Service (`WifiP2pDnsSdServiceInfo`)	Nearby Service Advertiser (`MCNearbyServiceAdvertiser`)
P2P Service Request (`WifiP2pServiceRequest`)	Nearby Service Browser (`MCNearbyServiceBrowser`)
Wi-Fi P2P Device (`WifiP2pDevice`)	Multipeer ID (`MCPeerID`)

3.3.1 Advertising available P2P services

Android:

Instantiate a P2P Service object. Start the `WifiP2PManager` and use `addLocalService` to add the P2P Service.

iOS:

Instantiate and initialise a Nearby Service Advertiser object.

3.3.2 Scanning for nearby devices and services

Android:

Call `discoverServices` on the `WifiP2PManager` with a `DnsSdTxtRecordListener` callback object. When a P2P Service is discovered, the `onDnsSdTxtRecordAvailable` callback function will be activated on that object.

iOS:

Instantiate a Nearby Service Browser object and call the `startBrowsingforPeers` function. The `foundPeer` callback function will be activated when a P2P Service is discovered.

3.3.3 Connecting to nearby devices and services

Android:

Once a device or P2P service is discovered, request to connect to it by calling `connect` on the WifiP2P Manager with the device MAC address.

iOS:

Once a device or P2P service is discovered, use the Nearby Service Browser `invitePeer` function to add the discovered device to the `MCSession`.

From the explanation above, we can see that both Android and iOS provide functionality for advertising and discovering P2P services, and connecting devices with shared P2P services. In the next subsection we will explain why we chose Android as our development platform.

3.3.4 Platform Selection

It was decided to select Android as the development platform for our framework because of the following reasons:

- Android is open-source, simplifying implementation and debugging.

- The Android Wi-Fi P2P implementation has been available since 2012, allowing insights to be gained from other Wi-Fi P2P projects, whereas iOS released its Wi-Fi P2P implementation later.
- Android OS has a much larger user-base than Apple iOS, providing the user with a large number of connection opportunities.

3.3.5 Android Wi-Fi P2P Constraints

In the previous subsection, we selected Android as our development platform. Before we continue to the framework design, it is important to consider the Android specific constraints.

3.3.5.1 Connection Acceptance Dialog

Android forces the user to authenticate a Wi-Fi P2P connection with a acceptance dialog or PIN. This means that at least one of the users will have to interact with their device to establish a connection.

3.3.5.2 Wi-Fi P2P Discoverable

In order to conserve energy, Android turns Wi-Fi P2P scanning off after 120 seconds. This also causes the device to become undiscoverable, because both devices need to be scanning to be discovered. This constraint forces the Wi-Fi P2P discovery to be restarted frequently to keep the device discoverable.

In this section we discussed what functionality mobile operating systems provide for establishing Wi-Fi P2P networks. Android and iOS provide basic functionality for advertising, discovering and connecting devices over Wi-Fi P2P. Android was chosen as the preferred platform and some platform specific constraints were mentioned.

3.4 Framework Design Choices

Using the knowledge and constraints of the previous sections, this section describes what design choices were made regarding user roles, connection automation and energy requirements.

3.4.1 Network Roles

According to the Android connection acceptance constraint, at least one of the users need to interact with their device to establish a connection. For this reason, it is recommended that users receiving content accept the connection, as they will also be looking at the screen. Other users in the network can then share content without device interaction. This gives rise to two different user

groups, based on their interactions, henceforth called **Consumer** users and **Provider** users.

Consumer users are so named, because they would like to receive content and use services from other devices. They are assumed to be stationary or at least actively interacting with the device. Although **Consumer** devices can also offer services, they are assumed to be primarily interested in services that other devices can offer.

In contrast to **Consumer** users, **Provider** users do not expect services to be offered to them at the moment, but assume that their generosity will be returned once they decide to start receiving content from other users.

It is important to note that Android uses different terminology when describing device roles in a network, namely Wi-Fi P2P group owners and Wi-Fi P2P clients. Wi-Fi P2P group owners are always responsible for hosting the Wi-Fi P2P network, hence if they leave, the network is disconnected. Wi-Fi P2P clients interact with the Wi-Fi P2P group owner similarly to a standard Wi-Fi client connecting to an access point.

Although **Consumer** users will mostly be Wi-Fi P2P group owners, it is also possible to be a **Consumer** and a Wi-Fi P2P client, if there is another **Consumer** already hosting a Wi-Fi P2P group. For this reason **Consumer**s and **Providers** are so named for how they interact with other devices on the network, instead of their specific network function.

3.4.2 Connection Automation

Provider users passively share the services to other devices by running the framework on the device, and authorising the sharing of selected services. The **Provider**'s device then autonomously interacts with **Consumer**s' devices to provide services to them.

3.4.3 Energy Use

According to Trifunovic et al [59], a Wi-Fi P2P GO consumes 231.92 mW and a Wi-Fi P2P Client consumes 49.75 mW. For this reason, the **Consumer** user should carry the burden of energy use as she is receiving services.

3.5 Framework Components

In this section we discuss the different components that form part of the Wi-Fi P2P framework. Using the functionality provided by Android, we design a framework which can handle the state of Wi-Fi P2P network connections, as well as abstract the Wi-Fi P2P interactions so that application developers can use Wi-Fi P2P without prior knowledge of its specific operation details.

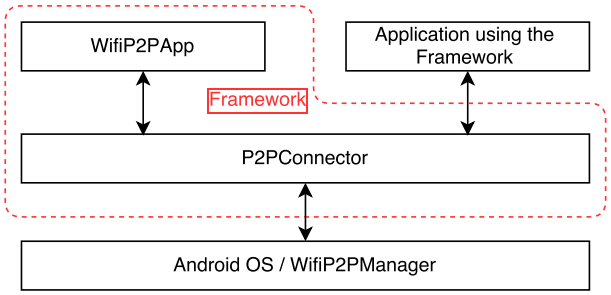

Figure 3.4: Wi-Fi P2P framework block diagram. The framework consists of the WifiP2PApp and P2PConnector, with the Android WifiP2PManager interface exposing Wi-Fi P2P functionality to the P2PConnector. Other applications can access the framework through an interface to the P2PConnector.

Figure 3.4 shows how the different components interact with each other. The framework consists of three distinct components which run on each device that has it installed: the Android Wi-Fi P2P manager (WifiP2PManager), the P2P connector agent (P2PConnector), and the control application (WifiP2P-App).

WifiP2PManager is provided by Android and it is the native interface for scanning for Wi-Fi P2P devices, facilitating connects and disconnects, registering P2P services, and discovering P2P services from other devices.

The P2PConnector is responsible for maintaining the state of the P2P framework and initiating connections and scans when necessary. P2PConnector queries WifiP2PManager for the Wi-Fi P2P connection state and obtains details of discovered P2P services from WifiP2PManager. As was mentioned above, **Consumer**s may want to asynchronously use P2P services hosted by **Provider**s, while the **Provider**s are not currently interacting with their devices. The P2PConnector is responsible for initiating P2P scans, P2P service advertising and connections from **Provider**s to **Consumer**s, to establish relevant P2P connections.

As an additional feature of the P2PConnector, it implements a communication channel (also running over Wi-Fi P2P) using TCP. This channel can be used for distributing messages and updates to all of the connected devices.

WifiP2PApp is the user interface (UI) with which the user controls P2PConnector. WifiP2PApp displays the Wi-Fi P2P connection state and user role, and interacts with the P2PConnector when the user wants to become a **Consumer** or a **Provider**.

3.5.1 Interfaces between framework components

The WifiP2PApp displays network information and controls the P2PConnector and therefore needs to receive updates from and send commands to the P2PCon-

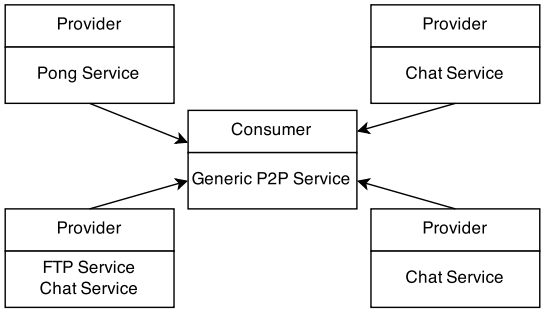

Figure 3.5: The Wi-Fi P2P environment. **Consumer** devices advertise a Generic P2P Service with a **Consumer** flag to indicate that they are willing to host a Wi-Fi P2P Group. **Provider** devices advertise other P2P services which might be relevant to the **Consumer** user.

nector.

The P2PConnector handles the state of the framework, listens to commands from WifiP2PApp and also sends commands to the WifiP2PManager.

Other applications needing to use the P2P framework can interact with the P2PConnector by remote procedure calling, according to the framework interface (discussed in detail in Section 4.2).

The WifiP2PManager reacts to commands from the P2PConnector and sends back updates of the network state.

3.5.2 Connection Procedure

By using the above mentioned components, the connection is established as follows:

Start Up

Figure 3.6 shows the sequence diagram of the start-up procedure of the Wi-Fi P2P framework:

1. If P2PConnector has not be been started before, it is started as soon as WifiP2PApp is opened. P2PConnector and WifiP2PManager run independently on the device.

2. P2PConnector starts a thread for advertising its P2P Generic Service with either a **Consumer** or **Provider** flag.

Connecting as Consumer

Figure 3.7 shows the sequence diagram of a **Consumer** receiving connections.

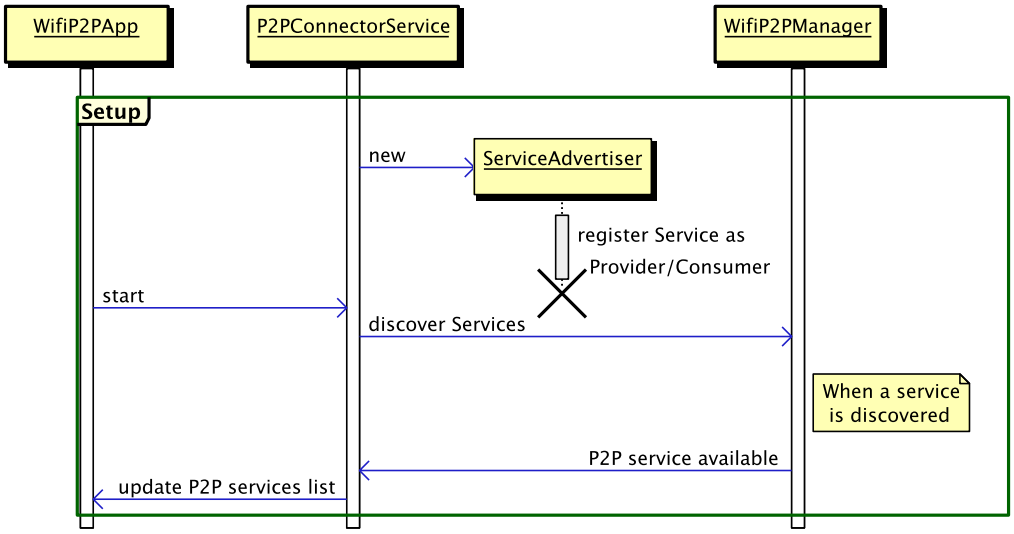

Figure 3.6: Start-up sequence of the P2PFramework showing the steps taken to start up the framework and prepare it for connecting to other devices.

1. After the P2P Generic Service has been registered, the **Consumer** device waits for **Provider** devices to connect to it.

2. When a **Provider** connects to this device, the connection status is updated to "Connected" in P2PConnector and WifiP2PApp.

3. P2PConnector starts an instance of ServerThread responsible for handling incoming TCP connections. ServerThread searches for an open port and starts listening on it. The port number is sent to P2PConnector which uses it to update the P2P Generic Service with the port number.

4. **Provider** devices then use the P2P Generic Service information to open a TCP connection to the **Consumer** on the specified port number. **Provider** devices test the connection by sending a initial test message and ServerThread responds to indicate that the connection is active.

Connecting as Provider

Figure 3.8 shows the sequence diagram of a **Provider** connecting.

1. **Provider** devices continually scan to detect **Consumer** devices in the vicinity.

2. Once a **Consumer** device is discovered, if this device can offer services that the **Consumer** is requesting, this device's P2PConnector requests WifiP2PManager to establish a connection to it.

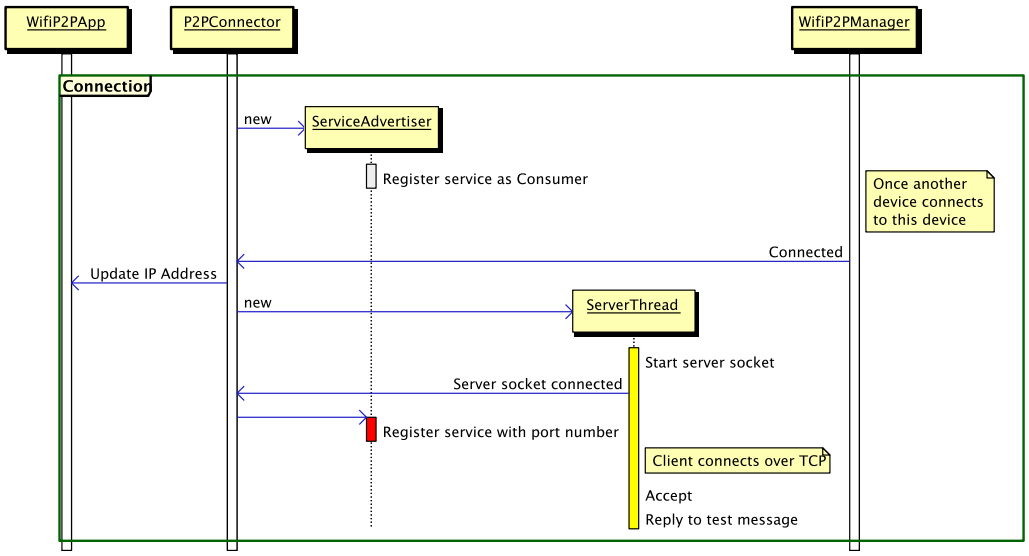

Figure 3.7: Steps taken by a **Consumer** device to establish a P2P connection with a TCP communication channel between all connected devices.

3. Once connected, WifiP2PManager sends connection information containing the P2P GO details to P2PConnector. P2PConnector restarts P2P discovery to ensure that the latest P2P Generic Service is discovered, which contains the TCP port number that ServerThread on the **Consumer** device is listening on.

4. Once the updated P2P Generic Service is discovered, P2PConnector starts a ClientThread instance to handle the outgoing TCP connection. ClientThread connects to the given IP address and port number to establish the TCP connection. ClientThread sends a test message to verify that the connection was successful.

5. Restart P2P services discovery. The Client-Thread also periodically (every 60 seconds) sends a keep alive packet to the Wi-Fi P2P GO to ensure that the TCP channel is ready for communication.

Disconnect Recovery

The framework provides two recovery mechanisms: reconnecting Wi-Fi P2P and reconnecting the TCP channel.

If a Wi-Fi P2P disconnect occurs, the P2P service discovery will once again discover the **Consumer** and initiate a connection according to Figure 3.8.

If the ClientThread does not receive a response from the Wi-Fi P2P GO within the time-out period, the ClientThread will use its connection IP and port number to re-establish the TCP connection.

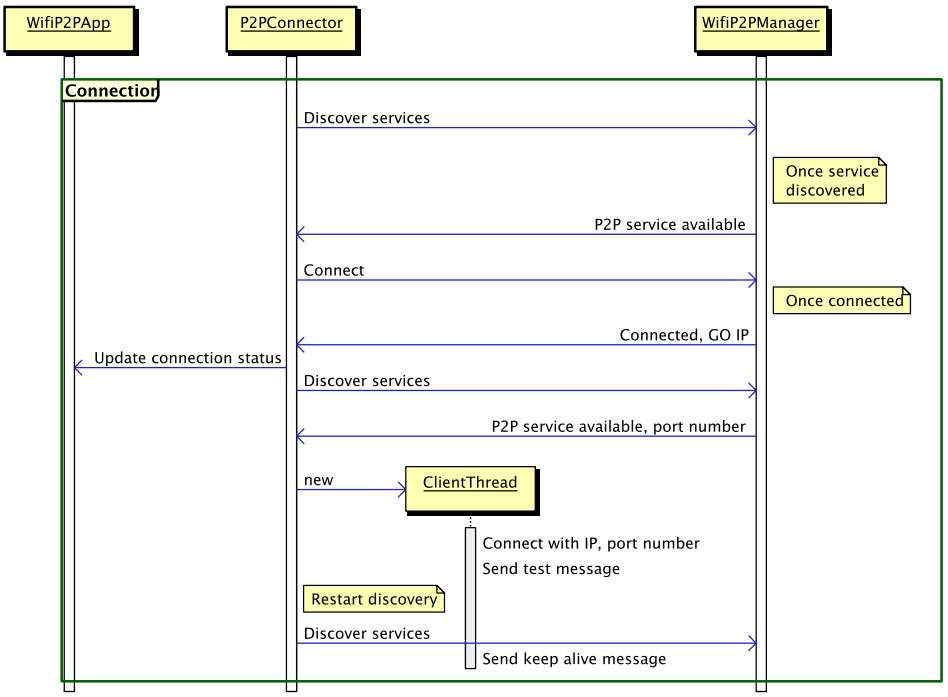

Figure 3.8: P2P Connect as **Provider**.

3.6 Using the Framework

Once the connection has been established, there are two ways of using the framework. As Wi-Fi P2P networks use IP addressing within the Wi-Fi P2P group (the same as standard Wi-Fi), packets can be sent to devices on the same Wi-Fi P2P group directly. As an alternative, the P2PConnector implements TCP communication between all connected devices, which can be used to multicast messages between devices.

The following subsections illustrate in two examples how the framework can be used. Detail implementations of these examples are given in Chapter 5.

3.6.1 Using IP Address: FTP Server

Android Wi-Fi P2P normally uses the 192.168.49/16 private IP address range, distinguishing itself from standard Wi-Fi communications which are assigned IP addresses according to the Wi-Fi access point's DHCP settings. The procedure for using the framework by IP addressing is explained in Figure 3.9 and below. The File Transfer Protocol (FTP) application is used as an example. The FTP application can be used to transfer files between connected devices. The functioning of the FTP server and client applications will be explained in more detail in Chapter 5.

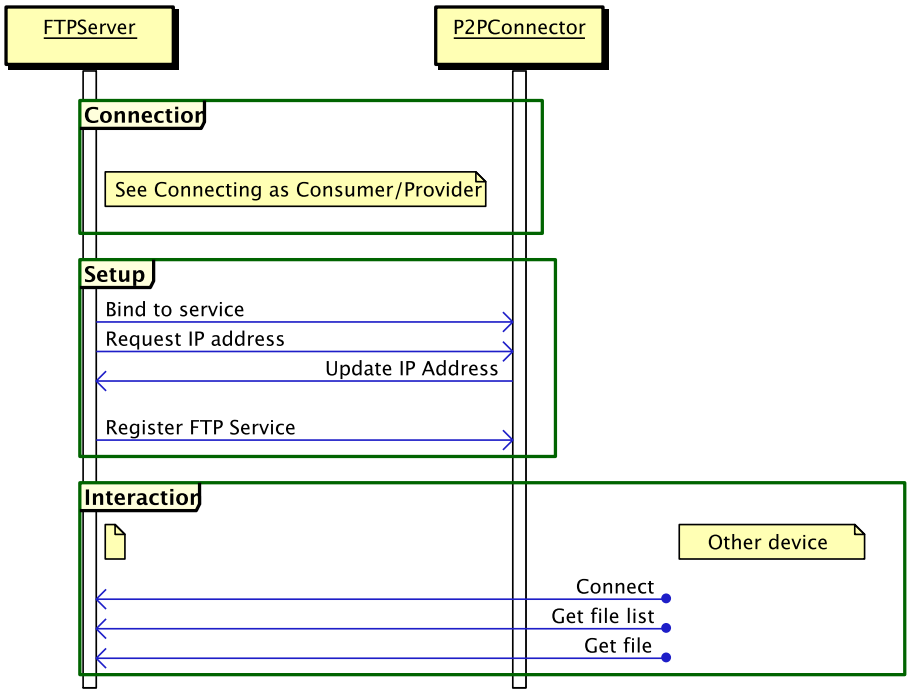

Figure 3.9: Using the framework via IP address.

Start Up

It is assumed that the devices have already been connected as explained in Section 3.5.2.

1. The FTP Server application binds to the P2PConnector and requests the P2P connection IP.
2. The FTP Server application starts the FTP service and requests that P2PConnector registers and advertises the FTP service at the given IP address and port number.
3. Devices interested in the FTP service can then use the P2P service information to connect to the given IP address and port. The FTP connections (using TCP as a transfer protocol) are made directly and packets are routed via IP, instead of being sent through the framework's TCP channel.

3.6.2 Using the General TCP Channel: P2P Chat

The P2PConnector also implements a TCP channel between all connected devices. Figure 3.10 shows how P2PChat interacts with the framework to establish a chat room with other devices.

Start Up

1. Each user interested in joining a chat room over Wi-Fi P2P, starts P2PChat application. P2PChat binds to the P2PConnector interface, i.e. gains access to the P2PConnector functionality. P2PConnector updates P2P-Chat on the Wi-Fi P2P connection state.

2. P2PChat requests that P2PConnector registers the Chat service, so that other devices can discover it.

Connecting

1. As this device's user is interested in receiving other users in a chat room, it acts as a **Consumer**. Wi-Fi P2P connections are established as described in Section 3.5.2. In accordance with the modular design principle, P2PConnector uses SocketSendDelegate (module responsible for sending P2P messages) and SocketReceiveDelegate (module responsible for receiving P2P messages) for handling TCP transmissions.

2. The user interacts with P2PChat through the UI to send a message. P2PChat attaches a Universally Unique Identifier (UUID) and sends the message to P2PConnector, which in turn forwards it to SocketSendDelegate for transmission.

Interaction

1. Incoming messages are received by SocketReceiveDelegate and is passed on to P2PConnector which routes it back to P2PChat to be displayed on the UI. P2P messages are filtered to display only chat messages, by checking the message UUID against the P2PChat UUID.

The ServerThread running on the **Consumer** device forwards all received packets to connected devices, ensuring that many-to-many communication is possible through the TCP channel. This connection medium is also used to update the P2P state between all connected devices, i.e. which devices are connected, and synchronise scanning as not to interfere with other Wi-Fi P2P operations.

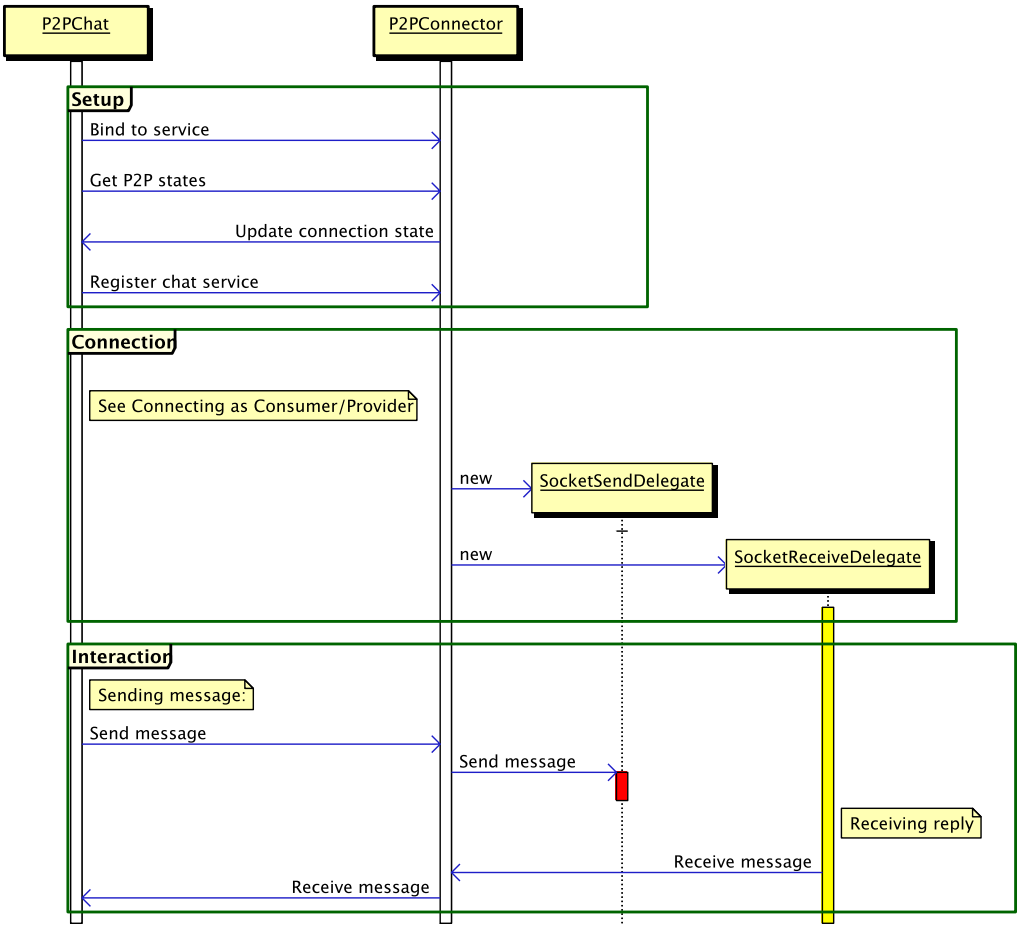

Figure 3.10: Using the framework and the generic TCP channel.

Chapter 4

Detail Design

In this chapter we describe the detail implementation of the proposed Wi-Fi P2P Framework as was set out in Chapter 3. The P2P framework is implemented using Java code on the Android Operating System (OS) as was described in Chapter 2.

In the this chapter and the next, we will use *WifiP2PApp* to indicate we are discussing a complete application, and `WifiP2PApp` to refer to a specific Java class.

4.1 Wi-Fi P2P Framework Implementation

In this section we will discuss the components which form part of Wi-Fi P2P framework and how they facilitate the interaction between user-orientated `Activitys`, other devices and the Android `WifiP2PManager`.

The Wi-Fi P2P framework is accessed using the AIDL interface `IP2PConnectorService`, shown in Figure 4.1.

The `P2PConnectorService` receives asynchronous Wi-Fi P2P state updates from the `WifiP2PManager` through the `P2PConnectorReceiver` and handles P2P service advertising and P2P connection requests. The `P2PWatchdogThread` is used to keep the `P2PConnectorService` alive and scanning for nearby devices, and resets the `P2PConnectorService` if an unrecoverable error occurs.

The `P2PConnectorService` can start a `ServerThread` or `ClientThread` for handling the general TCP channel. The `ServerThread`, running on the Consumer device, is responsible for handling incoming TCP connection requests and messages, forwarding messages to each connected device.

The `ClientThread` handles the TCP connection on the **Provider**.

Both the `ServerThread` and `ClientThread` dispatch messages with the aid of `SocketSendDelegate` and receive messages through the `SocketThreadedReceiveDelegate`. `SocketSendDelegate` and `SocketResponseThread` was implemented to be modular, so as to work in both the `ServerThread` and

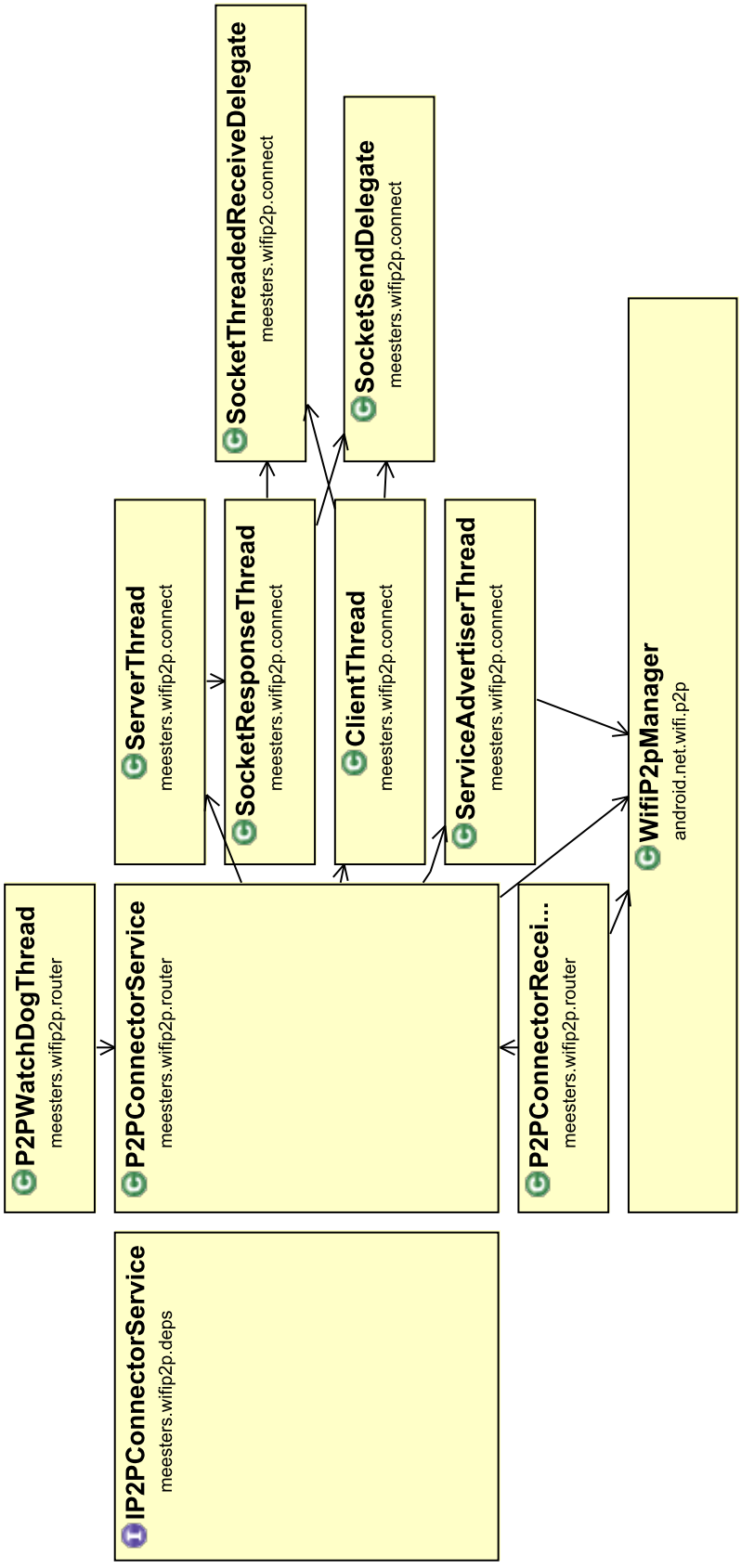

Figure 4.1: Wi-Fi P2P framework implementation block diagram.

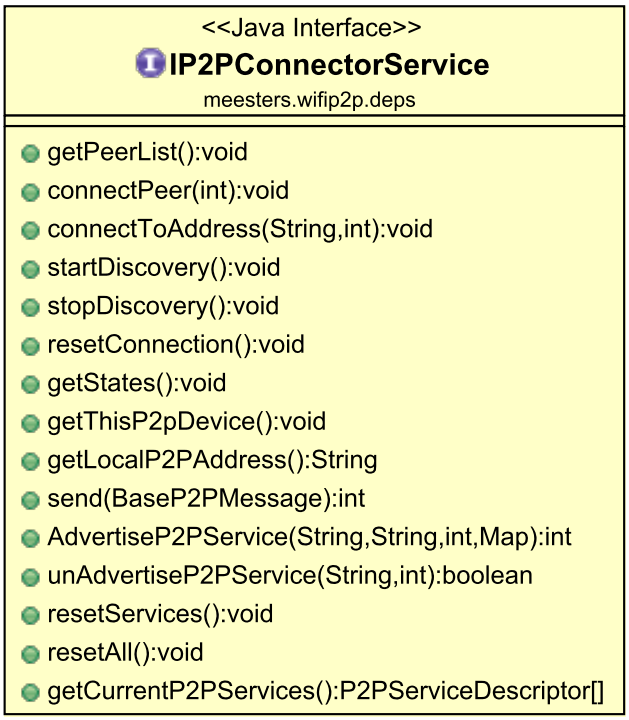

Figure 4.2: `IP2PConnectorService`: AIDL interface to the P2P framework.

`ClientThread`, assuring that both ends of the connection send and receive messages in the same way.

Each of the framework components is discussed in more detail in the following subsections.

4.1.1 Interface to the Framework: `IP2PConnectorService`

The P2P framework is accessed from `Activitys` using `IP2PConnectorService` interface. The interface exposes functionality for connecting to P2P devices, starting and stopping P2P service discovery, getting this device's Wi-Fi P2P MAC address and IP address, sending messages using the framework and for registering and removing advertised P2P services.

It is suggested that applications implement classes separating the UI code from the code to access the needed functionality of the `IP2PConnectorService`, according to the Bridge pattern.

4.1.2 Framework Core: `P2PConnectorService`

The `IP2PConnectorService` class is the interface for applications binding to the `P2PConnectorService`, and the functionality is implemented by `P2PConnectorService`.

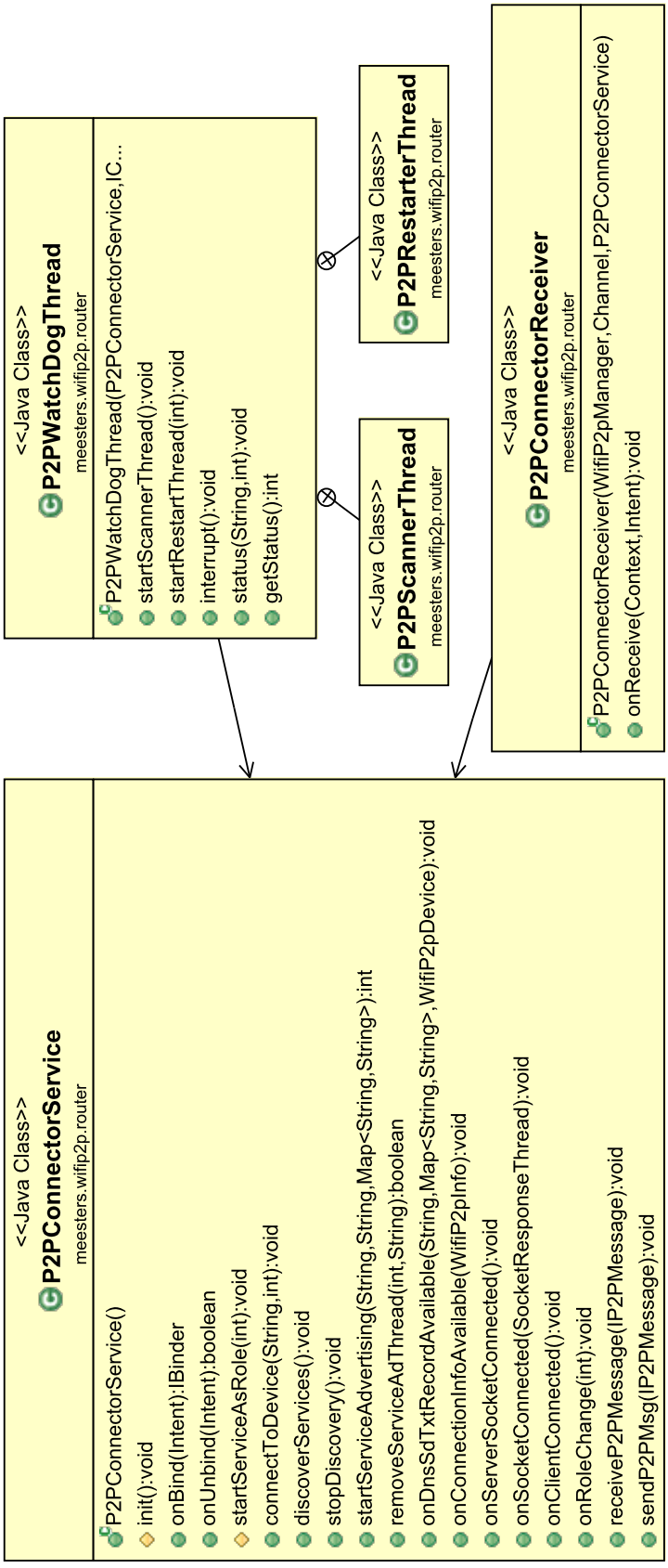

Figure 4.3: `P2PConnectorService`: The framework core process responsible for managing and starting all related Wi-Fi P2P processes.

Figure 4.3 shows the main functionality implemented in `P2PConnectorService`, the core process for managing and starting all related Wi-Fi P2P processes. `P2PConnectorService` has functionality which reacts to `Activity`s binding to the interface, starting and stopping P2P discovery, starting and stopping P2P service advertising, reacting to discovered P2P services, reacting to connecting TCP sockets and sending and receiving P2P Messages.

`P2PConnectorService` is designated as the hub of all information concerning the Wi-Fi P2P framework. It interacts with applications through `IP2PConnectorService`. Wi-Fi P2P state information is obtained by `P2PConnectorReceiver` listening for asynchronous Wi-Fi P2P state updates from the `WifiP2PManager`. `P2PConnectorService` also instantiates `ServerThread` and `ClientThread` depending on the device role.

Using the P2P framework, a typical connection procedure would proceed as follows (see also Figure 3.7 on page 23):

1. A user requesting to use the framework starts an `Activity` that uses the `IP2PConnectorService` interface. The Android OS binds the requesting `Activity` to the service through calling `onBind()` on `P2PConnectorService`.

2. Once bound, if this user acts as a Consumer, the `Activity` calls `onRoleChange()` with a Consumer parameter, causing `P2PConnectorService` to start a `ServiceAdvertiserThread` advertising its P2P Generic Service and thus signalling to other devices that it is ready for connections to it.

3. `WifiP2PManager` asynchronously updates Wi-Fi P2P state and calls `onConnectionInfoAvailable` on `P2PConnectorService` when a P2P device connects to this device. `P2PConnectorService` then starts a `ServerThread` to handle incoming TCP connections.

4. As soon as the `ServerThread` is ready, it calls `onServerSocketConnected` on `P2PConnectorService` to indicate that `P2PConnectorService` should update its P2P Generic Service with the correct port number of the `ServerThread`.

5. Once **Provider**s receive the updated P2P Service, they connect a TCP Socket to the port number specified in the P2P Service. `ServerThread` calls `onSocketConnected` to indicate that a client successfully connected.

6. If a message arrives on the TCP connection, `receiveP2PMessage` is called and `P2PConnectorService` broadcasts it to all listening `Activity`s.

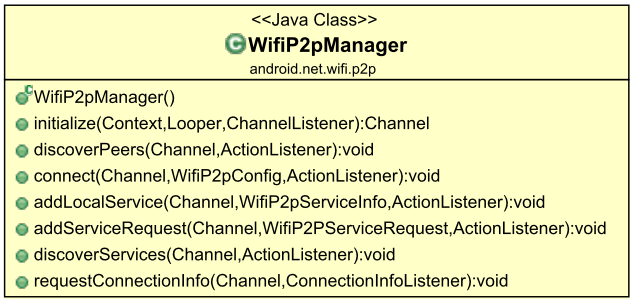

Figure 4.4: `WifiP2PManager`: The Android system service responsible for managing physical layer hardware of Wi-Fi P2P protocol.

4.1.3 Watchdog: `P2PWatchdogThread`

In order to ensure that `P2PConnectorService` remains responsive and the Wi-Fi P2P channel recovers from errors, `P2PWatchdogThread` runs two threads: `P2PScannerThread` and `P2PRestarterThread`.

`P2PScannerThread` is responsible for starting P2P service discovery every 30 seconds to ensure that devices discover each other, as both devices need to be scanning in order to discover P2P services and form P2P connections. Wi-Fi P2P scanning runs for a minimum of 60 seconds (depending on implementation). Restarting P2P scanning every 30 seconds ensures that devices are consistently discoverable and provides P2P service information to `P2PConnectorService` frequently, ensuring that all relevant services are discovered in a possibly dynamic environment.

Wi-Fi P2P service discovery is energy intensive [59], and frequent P2P service discovery will therefore consume large amounts of energy. Although this will ensure a frequent discovery of new P2P services, other models of service discovery timing could be considered to optimize energy usage of the P2P framework. An energy-optimized discovery model by Trifunovic et al [59] can be considered as a possible future improvement of this framework.

`P2PRestarterThread` is responsible for restarting the entire `P2PConnectorService` if an unrecoverable error happened. `P2PWatchdogThread` runs independently from the `P2PConnectorService` to oversee the restarting of the `Service`.

4.1.4 Physical Layer Manager: `WifiP2PManager`

`WifiP2PManager` shown in Figure 4.4 is responsible for handling physical layer interactions of the Wi-Fi P2P protocol. Before using `WifiP2PManager`, `initialize()` must be called to open a channel for communication.

`discoverServices()` was is used to discover devices with relevant P2P services. Before a discovery can be started, a `WifiP2PServiceRequest` specifying which P2P services to discover must be added to `WifiP2PManager`. Because

Figure 4.5: `ServerThread`: All the classes responsible for the TCP connection server-side.

the `P2PConnectorService` is interested in all available P2P services, a generic Service Request is added.

To connect to another P2P device, `connect()` is called with a `WifiP2pConfig` object containing the MAC address of the device to connect to. Each Wi-Fi P2P device can only belong to one Wi-Fi P2P group at a given time. Once connected, the connection information (for example IP address) can be obtained from `WifiP2PManager` by a call to `requestConnectionInfo()`.

P2P service advertising is done through `addLocalService()`.

4.1.5 Server Support Classes: `ServerThread`

The `ServerThread` instance is responsible for maintaining a TCP connection to every connected Wi-Fi P2P device. The device acting as the Consumer will instantiate `ServerThread`. This simplifies communications, as the Consumer device (and Wi-Fi P2P GO) will have an IP of 192.168.49.1 according to the Android Wi-Fi P2P implementation. Therefore all client devices will know which IP address to use, and obtain the port number from the advertised P2P Generic Service.

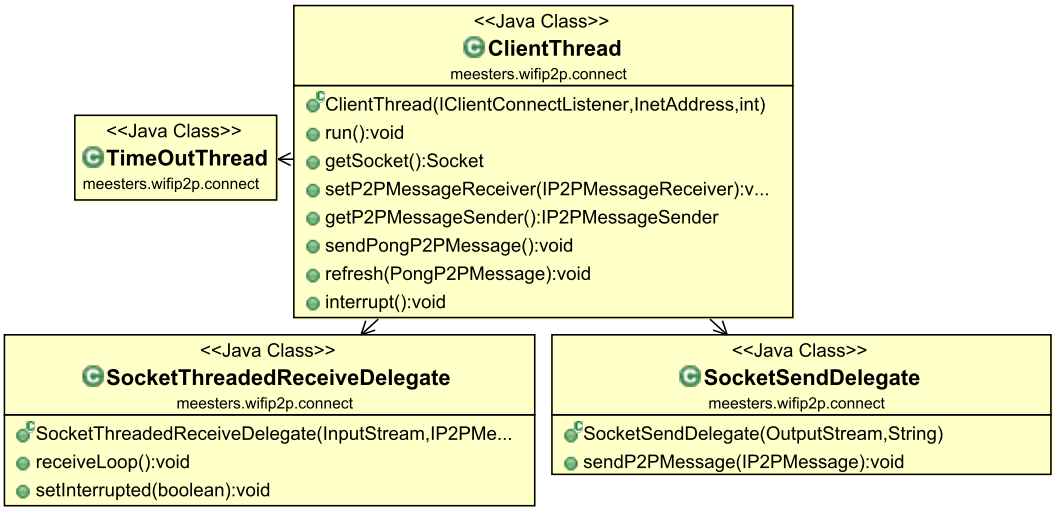

Figure 4.6: `ClientThread`: All the classes responsible for the TCP client-side.

For each incoming connection, a `SocketResponseThread` is instantiated. `ServerThread` keeps a list of all the `SocketResponseThread` and listens to the incoming messages from each. All devices in the Wi-Fi P2P network need to be able to send and receive P2P messages from all other devices in the network. `ServerThread` ensures that this condition is met by forwarding received messages to all `SocketResponseThread`s except the one that received the message.

Each `SocketResponseThread` retains an instance of `SocketThreadedReceiveDelegate` and `SocketSendDelegate`. `SocketSendDelegate` handles the sending of messages through the TCP channel and `SocketThreadedReceiveDelegate` handles the receiving of messages by continually listening for incoming messages on the TCP channel.

Each class of the three communication layers is responsible for a specific facet of the TCP connection: `ServerThread` handles incoming TCP connections and the routing of P2P messages to and from `P2PConnectorService` as well as between different `SocketResponseThread`s.

`SocketResponseThread` retains information about the Java `Socket` of its assigned connection and routes received P2P messages up to `ServerThread` and down to the `SocketSendDelegate`.

The `SocketThreadedReceiveDelegate` and `SocketSendDelegate` deal with the data streams associated with each TCP connection and translates P2P messages into objects sent across the network.

4.1.6 Client Support Classes: `ClientThread`

`P2PConnectorService` instantiates a `ClientThread` on Provider devices. The `ClientThread` stores information regarding the Java `Socket` and routes incom-

ing and outgoing messages for the `P2PConnectorService`. According to modular design principles, the `ClientThread` uses the same `SocketSendDelegate` and `SocketThreadedReceiveDelegate` as the `SocketResponseThread`. This ensures that the messages are sent and received in exactly the same way on both ends.

Similar to the `P2PConnectorService`, the `ClientThread` also retains an instance of a keep-alive thread, named `TimeOutThread`. `TimeOutThread` acts similar in some ways to the `P2PWatchdogThread` in keeping the TCP connection alive and restarting it if necessary. `TimeOutThread` is started as soon as the `ClientThread` connects. It then sends a keep alive packet every 60 seconds to check if the `Socket` is still open. If an answer packet is not received within another 60 seconds of the keep alive packet sending, it is presumed that the Socket has been closed due to unforeseen circumstances. The `TimeOutThread` then uses the original connection information to request a new connection from the other device.

4.1.7 P2P Framework controller application: *WifiP2PApp*

The *WifiP2PApp* application represents the UI front-end for monitoring and controlling the P2P framework.

The `WifiP2PApp` class receives Wi-Fi P2P connection state and P2P service updates through the `WifiP2PReceiver` (implementing the Android `BroadcastReceiver`). In addition to the connection state, `WifiP2PApp` also displays the P2P IP address, this device's Consumer or Provider state, current Wi-Fi P2P discovering state (actively discovering or stopped) and nearby P2P devices and services. `WifiP2PApp` also has UI buttons for starting and stopping P2P service discovery, and changing device role from Provider to Consumer.

`GenericP2PBridge` is implemented according to the Bridge pattern, and separates the UI code in `WifiP2PApp` class from the code to access the P2P framework, resulting in a modular application.

4.2 Summary

In this chapter we discussed the design patterns used in creating this framework. We then went on to discuss accessing the framework through the `IP2PConnectorService` interface. The interface exposes the functionality of `P2PConnectorService`, the main component of the framework, responsible for handling the state of the P2P Connection and starting other modules such as `ServerThread` and `ClientThread` responsible for maintaining the TCP connection between connected devices. We also discussed *WifiP2PApp*, the framework controller application.

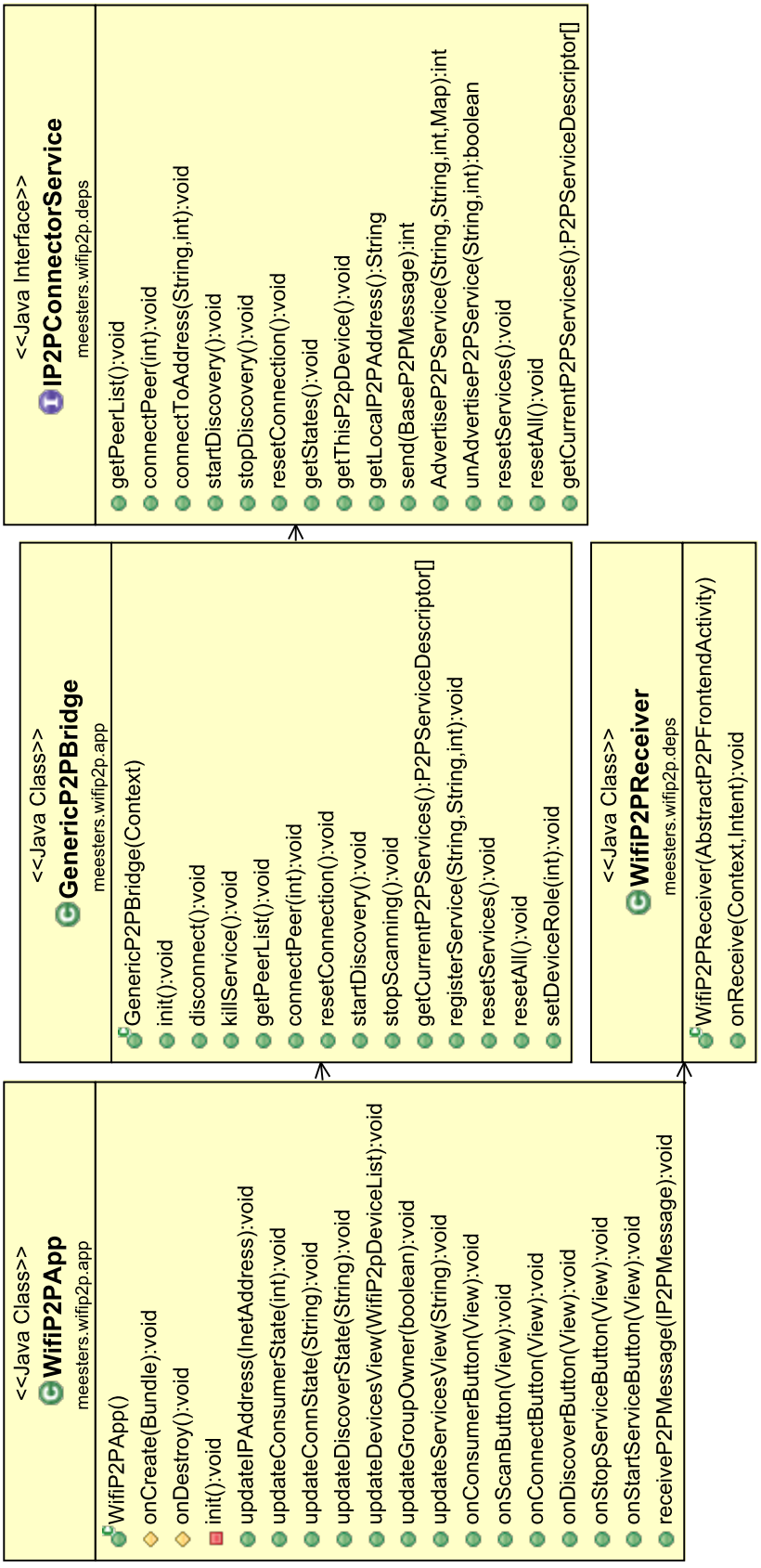

Figure 4.7: *WifiP2PApp* structure: All classes required by the *WifiP2PApp* to control the P2P framework.

In the next chapter we will discuss the implementation and adaption of four applications to use the Wi-Fi P2P framework.

Chapter 5

Application Implementation

The implementation of the applications enabling the use-cases mentioned in Section 3.1, are discussed here. As was mentioned in Section 3.1, each application tests a component of Wi-Fi P2P capabilities: content sharing will test the throughput, gaming will test latency, jitter and packet loss and chatting will test multiple device support.

Specific care will be taken to illustrate modular design principles and ways of using the framework.

5.1 Content Sharing using FTP Server and FTP Client

It was decided to implement content sharing using the File Transfer Protocol (FTP), which is widely used and supported. *Swiftp* is a open-source FTP Server application, available at [11]. The *Swiftp* application was adapted to accept connections on the Wi-Fi P2P interface as well as standard Wi-Fi, and to broadcast its presence as a Wi-Fi P2P Service. *WifiP2P_FTP*, the FTP Client application, was then developed using some of the *Swiftp* classes. *WifiP2P_FTP* measures the time taken to complete each download, used in calculating throughput. As was mentioned in Section 2.2.3, FTP uses Transfer Control Protocol (TCP), i.e. connection orientated channels for transferring files. Note that both FTP server and client establish TCP connections using the IP address obtained from the framework, instead of using the framework's generic TCP channel. Both application implementations are discussed in more detail in following subsections.

5.1.1 *Swiftp* App

The *Swiftp* application structure is shown in Figure 5.1.

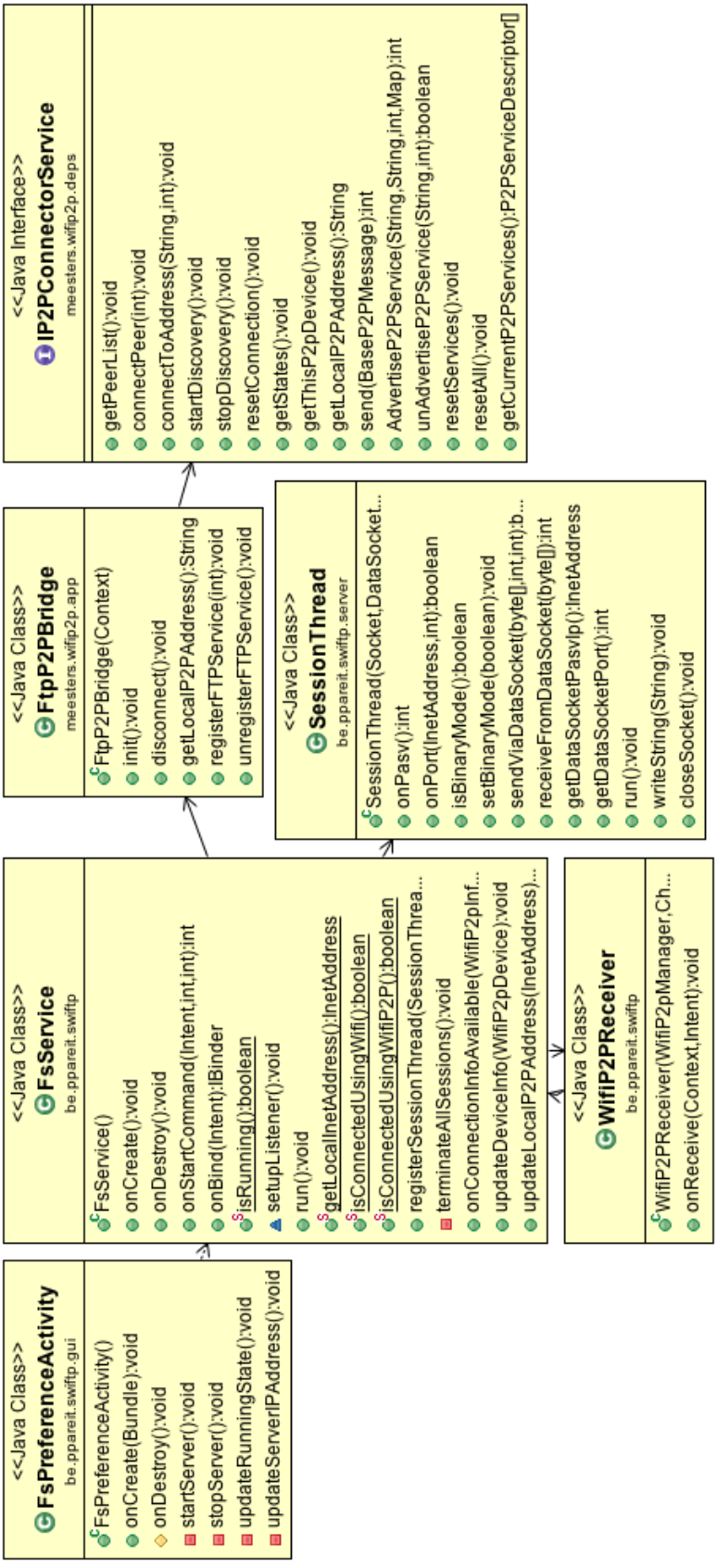

Figure 5.1: *Swiftp* application structure. `FsPreferenceActivity` represents the UI component and `FsService` the background process for handling incoming FTP connections. `WifiP2PReceiver` receives updates from the `P2PConnectorService` and `FtpP2PBridge` is used to call functions on the `IP2PConnectorService`.

The `FsPreferenceActivity` class, used as is from the open-source project, enables the user to start and stop the server and set the port number of FTP server. It also displays the running state and IP address of the FTP server.

`FsService` is a background process for managing the FTP server, listening for incoming TCP connections, starting a `SessionThread` for each connection, checking network connectivity, and updating the local IP address according to connection medium.

`FsService` was modified from the open-source project to include Wi-Fi P2P as a transfer medium, by responding to `WifiP2PReceiver` updates (indicating Wi-Fi P2P connected state and P2P IP address) and using the `FtpP2PBridge` to register and unregister the FTP P2P service.

The `SessionThread` class manages the FTP connection. `SessionThread` responds to FTP commands "PASV" (`onPasv()`), "PORT" (`onPort()`) and "TYPE" (`setBinaryMode()`), setting up the data connection in the correct mode. `SessionThread` also has functionality for writing FTP commands (`writeString()`), sending and receiving files through data connection, and closing the command and data sockets.

`FtpP2PBridge` was implemented according to the Bridge design pattern. The `FtpP2PBridge` binds to the `IP2PConnectorService` interface, separating the `FsService` code from the P2P framework, as well as hiding unnecessary functionality from the `FsService`.

Note that only small changes needed to be made to the `FsService` to support Wi-Fi P2P. Furthermore, `FtpP2PBridge` decouples the implementations so that `FsService` and `P2PConnectorService` can change independently.

Swiftp is not used to perform measurements, but is needed as FTP Server to transfer the files used for measuring.

5.1.2 *WifiP2P_FTP* App

Figure 5.2 shows the *WifiP2P_FTP* application structure. The `WifiP2P_FTP` class is the front-end activity giving the user control of the FTP client. `WifiP2P_FTP` is used to scan for P2P services, to connect to FTP servers, and to choose which files to download. `WifiP2P_FTP` starts the `SessionThread` to facilitate the connection to a FTP server. The `SessionThread` class was used from the *Swiftp* application, ensuring that the FTP server and client implementation mirror each other.

`WifiP2PReceiver` is used for reporting P2P FTP services, and the P2P discovery is started through `WifiP2P_FTPBridge`.

`WifiP2P_FTP` measures the throughput by logging download start time and finish times to a log file. The throughput is then calculated by applying equation 2.3.1.

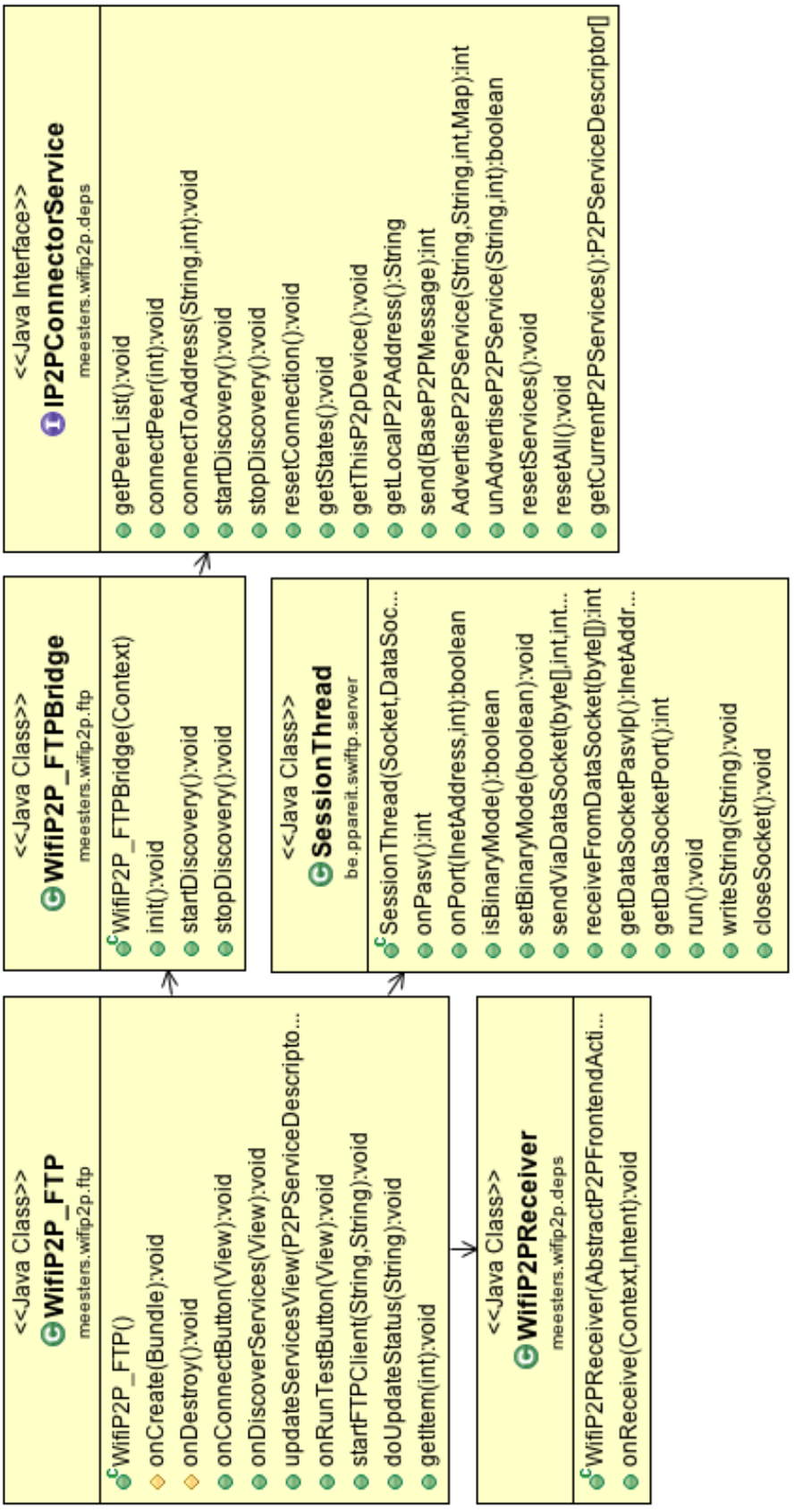

Figure 5.2: *WifiP2P_FTP* application structure. `WifiP2P_FTP` enables the user to browse and download files from the FTP server. `SessionThread` is started by `WifiP2P_FTP` when the FTP client connects to the FTP server, handling FTP commands and facilitating file download. `WifiP2PReceiver` receives updates from the `P2PConnectorService` and `WifiP2P_FTPBridge` is used to call functions on the `IP2PConnectorServiice` for starting and stopping P2P service discovery.

5.2 Gaming Using the *PongWifiP2P* App

Pong is a game application simulating table-tennis between two players, each controlling a paddle to try and stop the ball passing them. The *PongWifiP2P* application is based on an open-source project [33], and was also adapted for Wi-Fi P2P. This game application was chosen as it is latency sensitive and packet loss would be clearly evident (seen as the ball jumping irregularly).

One of the devices acts as the master (host), calculating and updating the game state, while the other (slave) device receives continuous updates and controls the opposite paddle.

The *PongWifiP2P* application structure is shown in Figure 5.3.

The `Pong` activity class displays the `PongView` and was adapted to use the `PongP2PBridge` to access the P2P framework. `Pong` receives Wi-Fi P2P connection state and P2P service updates from `WifiP2PReceiver`.

`PongP2PBridge` was implemented according to the Bridge pattern, and enables `Pong` to get the P2P IP address, register and unregister the Pong P2P service, start and stop P2P discovery, and request P2P Pong game services.

The `PongView` class displays the game view and updates the game state. If this device is the master, it uses `sendGameState()` to send the position of the ball and the blue paddle using `updateVariablesServer()` in the `PongController` class. If this device is the slave, it only sends the position of the red paddle.

`PongController` was implemented to use UDP packets to send the game state, a packet number and timestamp to the connected device. On the receiving end, `PongController` replies to each successfully received packet using `sendDebugMessage()`. The reply packet contains the original packet number and timestamp, which is used to calculate the round-trip latency of each packet at the original sender's side. Unsuccessfully received packets (i.e. CRC check failed) are logged as packets lost.

`PongClientThread` receives incoming UDP packets, updating the `PongView` with the latest game state.

Using `doAI()` in the `PongView` class, the game was played by AI characters to prevent human irregularity to interfere with testing.

To clarify, `Pong` and `PongView` were adapted from the open-source project, and `PongClientThread`, `PongController`, `PongP2PBridge`, and `WifiP2PReceiver` were implemented.

5.3 Chatting Using the *WifiP2P_Chat* App

The *WifiP2P_Chat* application was developed to test chatting between multiple devices. Figure 5.4 shows the application structure.

The UI class, `WifiP2P_Chat`, shows the Wi-Fi P2P connected state and allows the user to enter and send text messages.

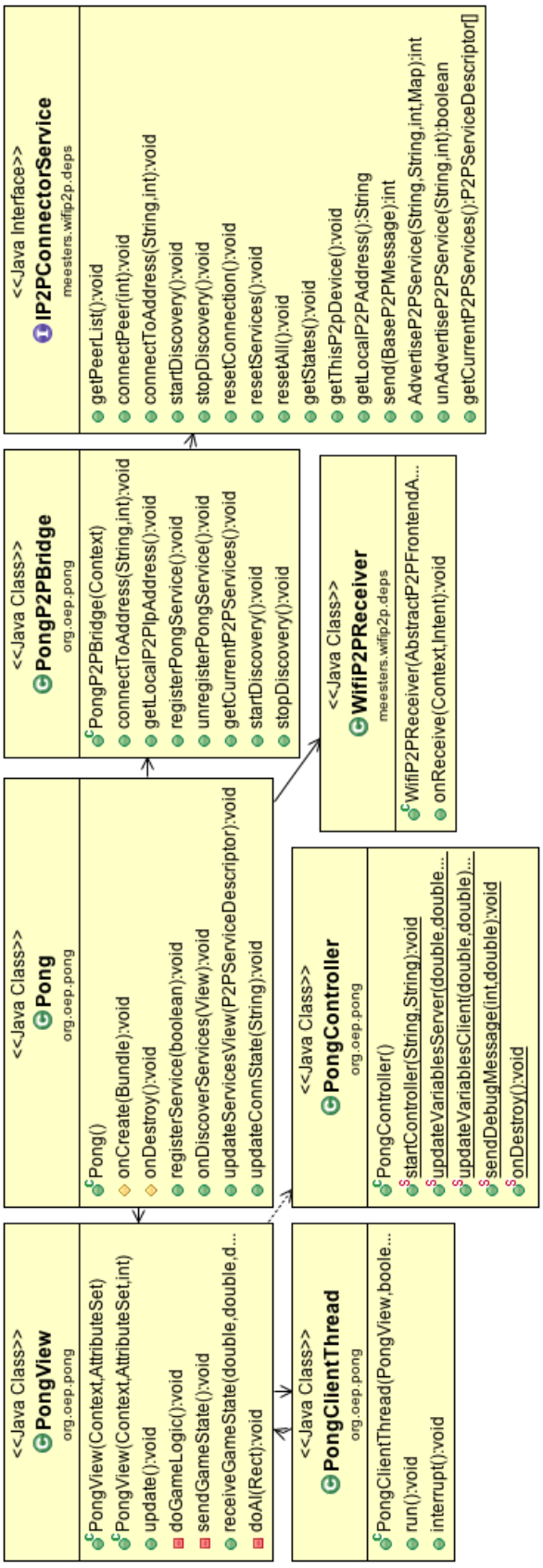

Figure 5.3: *PongWifiP2P* application structure. The `Pong` class displays `PongView`, uses `PongP2PBridge` to access the P2P framework and receives P2P updates from `WifiP2PReceiver`. `PongView` displays the game, updates the game state, sends the game state using `PongController` and receives game state updates through `PongClientThread`.

For sending text messages, `WifiP2P_Chat` calls `sendMessageToPeers()` on `P2PChatBridge`. `P2PChatBridge` translates this message into a `P2PMessage` and sends it to the P2P framework.

The `P2PMessage`, designed according to the Flyweight pattern, represents a message object in the P2P environment. `P2PMessage`s contain a Universally Unique Identifier (UUID), message type, origin device MAC address and data. The UUID is used to identify which application sent this message, `WifiP2P_Chat` in this case.

Using `P2PMessage`, the P2P framework can send this message to all connected devices in the generic TCP channel. On the receiving devices, `WifiP2P-Chat` distinguishes chat messages between other `P2PMessage`s by checking the UUID.

`WifiP2PChatReceiver` receives Wi-Fi P2P connection state updates and `P2PMessage`s.

5.4 Modular Design

As can be seen from the previous application implementations, only minor changes needed to be made to the existing applications. Using the Bridge pattern also allows application and P2P framework implementations to change independently. It can also be noted that `WifiP2PReceiver` is reused in all three applications without changing the implementation.

5.5 Using the Framework

As was mentioned in Chapter 3, there are two ways of accessing the framework: using the Wi-Fi P2P IP or by using the generic TCP channel. Both cases, with their advantages and disadvantages, are discussed here with reference to the above mentioned applications.

5.5.1 Using the P2P IP Address: *WifiP2P_FTP*

The *WifiP2P_FTP* application uses the IP address of the Wi-Fi P2P interface and establishes TCP connections as needed (see Figure 3.9 on page 25). This allows for parallel connections (using port numbers) between devices, which is needed for FTP. However, by forming their own connections, application connections are not aware of P2P network interrupts, and thus are not able to recover as easily.

5.5.2 Using the Generic TCP channel: *WifiP2P_Chat*

The *WifiP2P_Chat* application uses the generic TCP channel by sending `P2PMessage`s to the P2P framework (see Figure 3.10 on page 27. The frame-

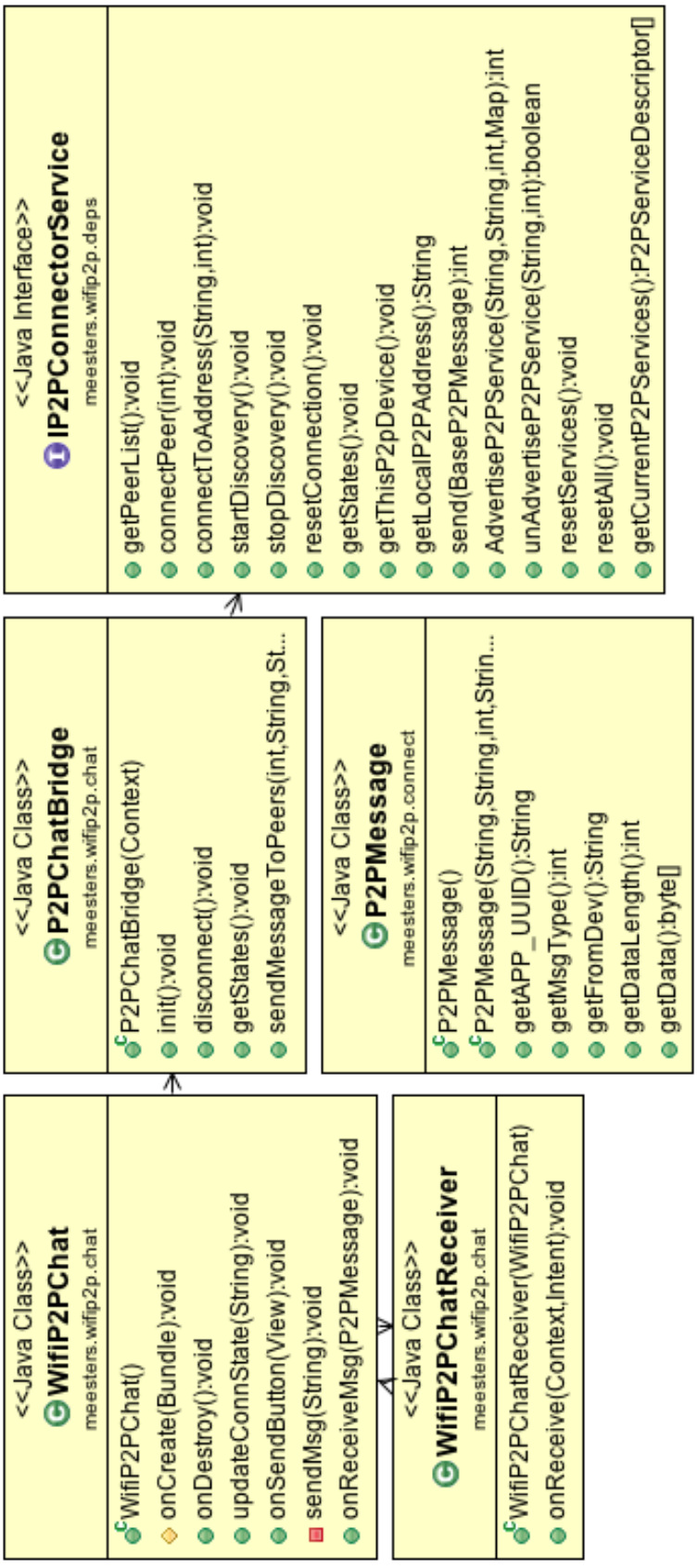

Figure 5.4: The `WifiP2PChat` class allows the user to view, enter and send text messages. `P2PChatBridge` sends the message as a `P2PMessage` using the P2P framework generic TCP channel. `WifiP2P_Chat` receives updates of P2P connection state and `P2PMessages` through `WifiP2PReceiver`.

work then dispatches these objects using the `ClientThread` and `SocketSendDelegate`.

Using the generic TCP channel is advantageous because it does not require prior knowledge of Wi-Fi P2P and does not require any connectivity management. However, messages need to be packed into the `P2PMessage` object and also if the generic TCP channel is busy, messages could become delayed.

5.6 Summary

In this chapter we discussed the implementation of applications for the use-cases mentioned in Chapter 3. *Swiftp* and *WifiP2P_FTP* are used together to download files over FTP, *PongWifiP2P* is used to play Pong between two devices and *WifiP2P_Chat* enables chatting between multiple devices.

We also briefly discussed how the framework can be used. The Wi-Fi P2P framework generic TCP channel was used by *WifiP2P_Chat*, translating user text messages to `P2PMessage`s and letting the P2P framework dispatch the message to all devices. Alternatively, the P2P framework IP address was used by *WifiP2P_FTP* and TCP connections were formed by the application itself.

In the next chapter we will discuss the tests executed and results obtained using the Wi-Fi P2P framework and the applications implemented.

Chapter 6

Testing

In the previous chapter, we explained which applications were implemented to satisfy the use-cases mentioned in Section 3.1. In this chapter we use the applications implemented to measure different aspects of Wi-Fi P2P and evaluate its suitability to each of the use-cases.

Firstly, we explain which Android devices were used for testing and how we evaluated test sample sizes. We then give results measured for throughput, latency, packet loss and multiple device support. Throughput, latency and packet loss results are compared to standard Wi-Fi, and throughput tests were also carried out using Bluetooth.

Android's implementation of Bluetooth is limited to TCP communication, and therefore latency, jitter and packet loss of such a channel would not be an applicable comparison to UDP latency, jitter and packet loss.

Energy usage was measured during Bluetooth, Wi-Fi and Wi-Fi P2P service scanning, connected and downloading over FTP.

Finally, P2P connection time and P2P connection success rate was measured.

6.1 Testing Devices

All tests were run on three devices (chosen as a representative sample of the low-to-medium performance Android devices): a Samsung S3 Mini, a Vodafone Smartmini 4 and a Google (Asus) Nexus 7 (2012). The different device results were aggregated to give a distribution representative of all three devices. To obtain a lower bound for performance metrics, the device with lower CPU power hosted the processing-intensive services.

For clarity the Samsung S3 Mini is referred to as Samsung Mini and the Vodafone Smart 4 mini as Smart4. Table 6.1 shows a summary of device specifications.

Table 6.1: Device technical specifications summary.

	Samsung S3 Mini	Vodafone Smart4 Mini	Google (Asus) Nexus 7 (2012)
CPU Speed [MHz]	1000	1300	1200
CPU Type	2 Core Coretex-A9	2 Core Cortex-A7	4 Core Cortex-A9
RAM [MB]	1024	512	1024
Release Year	2012	2014	2012

6.2 Confidence Interval on the Mean

When working with sampled data, calculating the confidence interval on the mean of the sampled data gives insight into the reliability of the sample dataset. The 95% confidence interval on the sample mean signifies that we are 95% confident that the true mean of the measured phenomenon lies within the confidence interval [7]. A Student's t distribution [30] (instead of the normal distribution) is used as the variance of the sample set is not known, but rather estimated from the sampled data. The confidence interval on the mean is calculated as follows:

First, we need to calculate the estimate of standard error s_M

$$s_M = \frac{s}{\sqrt{N}} \tag{6.2.1}$$

where s is the sample standard deviation and N is the number of samples in the measured data. Using the equation above, we compute the lower and upper limits of the 95% confidence interval:

$$\text{Lower bound} = \bar{x} - s_M \times t_{95\%} \tag{6.2.2}$$

and

$$\text{Upper bound} = \bar{x} + s_M \times t_{95\%} \tag{6.2.3}$$

where $\bar{x}$ is the mean of the samples, and $t_{95\%}$ is the value from the Student's t distribution [30] for a given sample size. The difference between these two bounds is called the Margin of Error (ME):

$$\text{ME} = \text{upper bound} - \text{lower bound} = 2 \times s_M \times t_{95\%} \tag{6.2.4}$$

The ME is used in this project as a metric of good sample size. As a rule of thumb, a ME of less than 10% of the sample mean value is accepted. That is, the mean of the sample set is within $\pm 10\%$ of the true mean of the phenomenon.

We will now continue to discuss the application tests.

6.3 Testing of Throughput

High throughput provides a better user experience for content sharing applications as more content can be accessed in a smaller amount of time. The *Swiftp*

and *WifiP2P_FTP* applications will be used to measure throughput on Wi-Fi P2P and standard Wi-Fi.

6.3.1 Wi-Fi P2P and Wi-Fi Throughput

Wi-Fi P2P can function without a fixed access point, but in some cases a Wi-Fi access point is available. For this reason and using Wi-Fi as a baseline, we compare Wi-Fi P2P and standard Wi-Fi, to evaluate the advantages of using Wi-Fi P2P for content sharing compared to standard Wi-Fi.

Test Setup

The *WifiP2P_FTP* application measures the Wi-Fi P2P throughput in megabytes per second (MB/s) by dividing file size by download time. The file transmission throughput was obtained by following these steps:

1. Connect two devices using *WifiP2PApp* for Wi-Fi P2P or connect to a Wi-Fi Access Point (AP) for standard Wi-Fi.
2. Start the FTP server on the weaker device. The FTP server will listen on all interfaces, including Wi-Fi P2P and Wi-Fi. The server device will act as the Provider.
3. Connect to the FTP server from the Consumer device with *WifiP2P-FTP*. Use the 192.168.49/16 address (shown in *WifiP2PApp*) when using Wi-Fi P2P or the Wi-Fi AP addresses when using Wi-Fi.
4. Download the 100 MB, 50 MB, 10 MB, 5 MB and 1 MB files repeatedly, in each case until 100 MB has been downloaded in total.
5. Log the download start time and end time and use the download time of each test to calculate the average throughput and throughput distribution over 100 MB.
6. Perform this process for the devices at 0 m, 5 m and 10 m apart. For standard Wi-Fi, one device remains next to the Wi-Fi AP, and the other device is placed at 0 m, 5 m and 10m.
7. Measure the Signal-to-Noise Ratio (SNR) of at every distance.

Measuring the SNR was completed by following these steps:

1. Connect to either Wi-Fi P2P group or Wi-Fi AP.
2. Open *WifiSNR* [40] application, downloaded for this purpose. Let the app measure 60 seconds of SNR values on each device.
3. Average the values between the three device pairs.

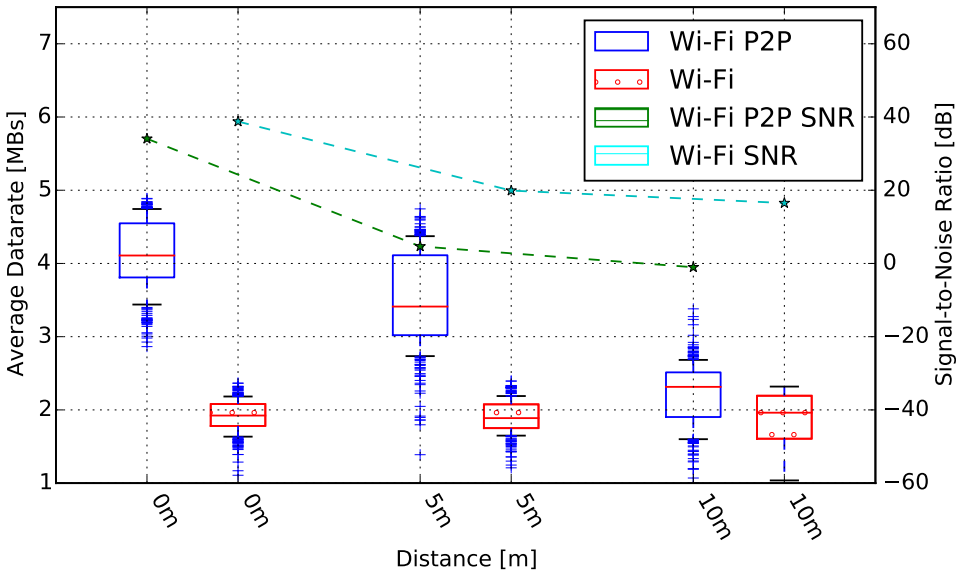

Figure 6.1: FTP throughput comparison between Wi-Fi P2P and Wi-Fi. Red lines indicate the median of measured data, box edges show the 25th and 75th percentile and box caps indicate 10th and 90th percentiles. Plus signs indicate outliers. Wi-Fi P2P and Wi-Fi Signal-to-Noise ratios are also indicated by dotted lines, with the dB scale to the right of the plot.

Expectation

It is expected that the Wi-Fi P2P and standard Wi-Fi bandwidth will be similar for 0 m. Due the limited transmitting power of a battery powered smartphone, SNR is expected to decrease as distance between devices increases. Lower SNR causes more transmission errors and thus more retransmit delays are expected over Wi-Fi P2P, resulting in standard Wi-Fi outperforming Wi-Fi P2P as device separation increases.

Results

Table 6.2 shows the average SNR at the given distances. The SNR of Wi-Fi P2P is always lower than that of standard Wi-Fi, as the mobile device has limited transmitting power. As the distance increases, the SNR also decreases, with Wi-Fi P2P SNR decreasing faster than standard Wi-Fi.

According to Figure 6.1, Wi-Fi P2P throughput is much higher than standard Wi-Fi at small distances, but drops steadily, to close to standard Wi-Fi for

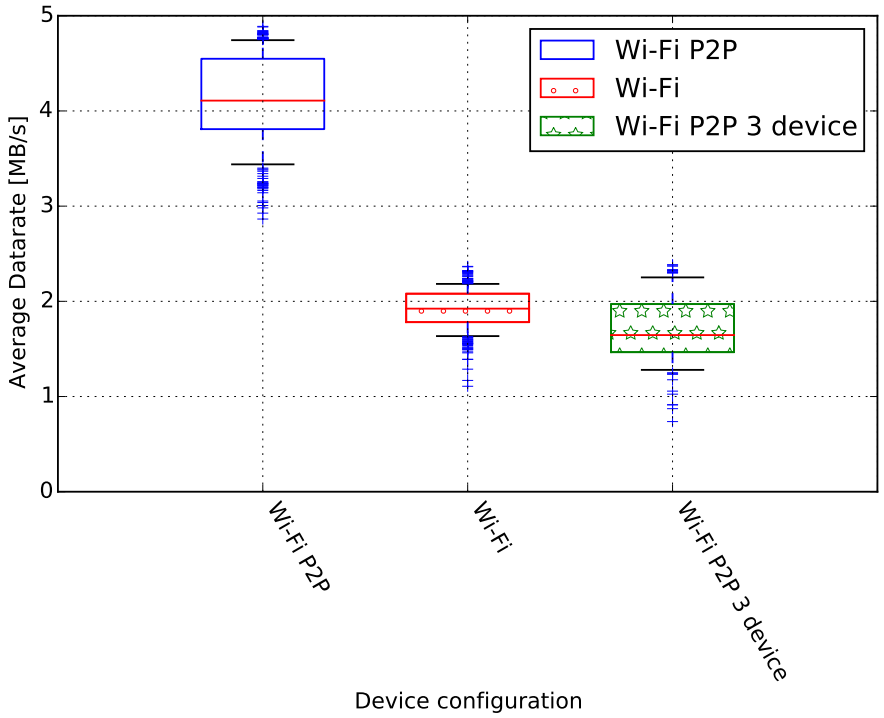

Figure 6.2: FTP throughput comparison between Wi-Fi P2P, Wi-Fi and Wi-Fi P2P in a three device configuration. Red lines indicate the median of measured data, box edges show the 25th and 75th percentile and box caps indicate 10th and 90th percentiles. Plus signs indicate outliers.

Table 6.2: Wi-Fi P2P and Wi-Fi Signal-to-Noise Ratios in decibels (dB), averaged between the three devices. Distances for FTP test.

	0 m	5 m	10 m
Wi-Fi P2P	34.07	4.67	-1.03
Wi-Fi	38.77	19.90	16.50

Table 6.3: Summary of Wi-Fi P2P and Wi-Fi throughput mean and ME.

Distance [m]	Mean [MB/s]	Margin of Error[MB/s]
0 m Wi-Fi P2P	4.105	0.110
0 m Wi-Fi	1.912	0.050
5 m Wi-Fi P2P	3.496	0.147
5 m Wi-Fi	1.906	0.051
10 m Wi-Fi P2P	2.215	0.098
10 m Wi-Fi	1.895	0.075

larger distances. The throughput of standard Wi-Fi remains roughly constant as the device separation increases, while the Wi-Fi P2P throughput steadily decreases.

The tests were run 20 times for each device pair, for a total of 60 tests. The throughput mean and ME are summarised in Table 6.3.1.

Discussion

Against expectation, Wi-Fi P2P throughput at small distances was much more than that of the standard Wi-Fi.

This result can be explained by the shorter "transfer path" between Wi-Fi P2P compared to Wi-Fi, i.e. standard Wi-Fi needs to send data via a router, while Wi-Fi P2P creates a direct connection between devices.

To verify this reasoning, a short experiment was conducted at 0 m, where Wi-Fi P2P throughput was measured using three devices, and downloading between the two "outer" devices, i.e. the Provider and Consumer were both Wi-Fi P2P Clients connected to a Wi-Fi P2P Group Owner (GO). This test result is shown in Figure 6.2. In this configuration Wi-Fi P2P throughput results are similar to standard Wi-Fi, corresponding to the explanation above.

The FTP throughput tests were not completed with distances larger than 10 m, as many TCP connection failures were already occuring at 10 m on Wi-Fi P2P. It is likely that the SNR is becoming so low and the packet loss high, so that a connection-orientated protocol such as TCP struggles to maintain reliability of the data-stream, and thus the congestion from packet resends make the channel unusable. Maximum range of Wi-Fi P2P is tested using the UDP protocol in the Pong tests (Section 6.4).

6.3.2 Bluetooth throughput

In addition to Wi-Fi P2P or Wi-Fi capabilities, most smartphones are equipped with Bluetooth. Thus, as an alternative device-to-device communications method, the throughput of Bluetooth was also investigated.

Test Setup

Android's implementation of Bluetooth does not use IPs to route packets, but rather transmits in a single serial communications connection. This resulted in the *WifiP2P_FTP* application not working over Bluetooth, as it uses IPs to connect. Instead, a Bluetooth chat example application [6] from Android was configured to act as a FTP server or client, and respond to standard FTP text commands. As far as possible, the same classes were used as for the *WifiP2P_FTP* application to enable an effective comparison between Wi-Fi and Bluetooth performance.

The throughput tests were completed as follows:

1. Connect two devices using the *BluetoothFTPChat* application.
2. Send the FTP command to retrieve a file, for 1 MB, 5 MB, 10 MB, 50 MB and 100 MB files and log the download start times.
3. Receive the files through the serial channel (the only channel available).
4. Once the end of stream character has been found, log the download finish times.
5. Perform these steps at 0 m, 5 m and 10 m devices separation.

Expectation

A throughput of between 1 and 3 Mbps (125 kB/s - 375 kB/s), as the highest supported Bluetooth version by all three devices is Bluetooth V3.0 without High Speed (HS) support.

It is also expected that the throughput of Bluetooth will decrease as the distance between devices increase.

Results

Figure 6.3 shows the results for the Bluetooth throughput test and as expected the throughput decreases as distance between devices increases. Variance of Bluetooth throughput results for 5 m and 10 m is unexpectedly small, while variance at 0 m is very large. Table 6.4 gives a summary of mean throughput and margins or error.

Discussion

The throughput measured in this test matches expectation, as Bluetooth V3.0 without High Speed support is limited to 375 kB/s. Also according to expectation, the throughput decreases as the distance between devices increases.

Unlike Wi-Fi P2P, Bluetooth is less sensitive to distance, by only dropping slightly as device separation increases. This could possibly be explained by

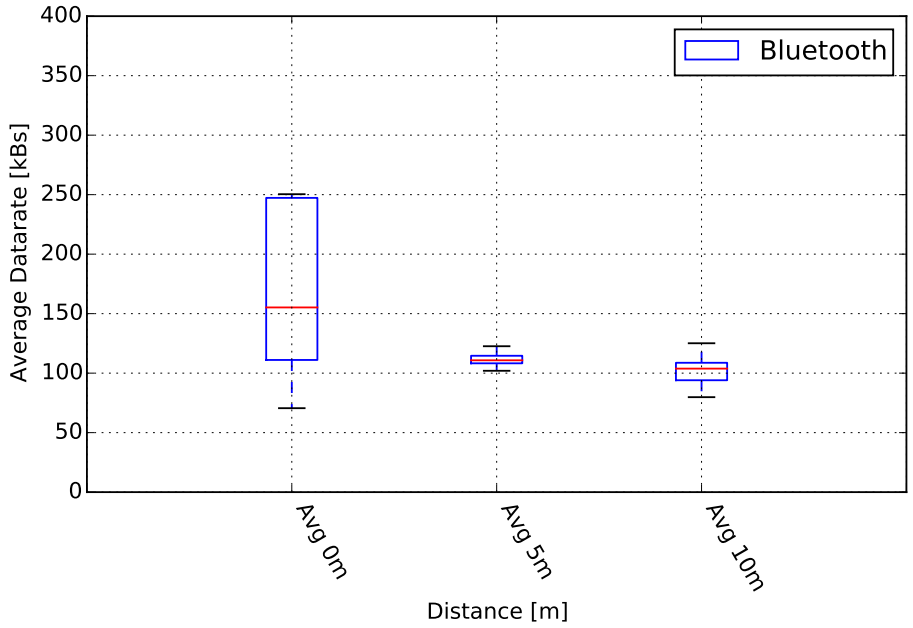

Figure 6.3: Bluetooth FTP throughput. Red lines indicate the median of measured data, box edges show the 25th and 75th percentile and box caps indicate 10th and 90th percentiles. Outliers beyond 10th and 90th percentile are not shown.

Bluetooth's lower data-rate constrained by energy-efficient protocol timings instead of bandwidth capacity. The small variance in Bluetooth throughput for 5 m and 10 m could also be explained by protocol timings constraining bandwidth use. The reason for the large variance at 0 m is unclear, and further tests should be conducted to verify this result.

The Bluetooth throughput is not displayed alongside Wi-Fi P2P results because of the order of magnitude difference. For instance at 0 m, the average throughput for Wi-Fi P2P is 4105 kB/s and Bluetooth is only 160 kB/s. As can be seen from these results, Bluetooth is not suited to transferring large amounts of data, but should rather be used in long-running energy sensitive tasks. Bluetooth V3.0 with High Speed support could provide better results: up to 3 MB/s theoretical [50] throughput, but none of the test devices could support this version of the Bluetooth protocol.

Table 6.4: Bluetooth throughput mean and margin of error.

Distance [m]	Mean [kB/s]	Margin of Error [kB/s]
0 m	160.503	15.025
5 m	111.865	6.092
10 m	102.985	6.777

6.4 Testing of Latency, Jitter and Packet loss

Latency, jitter and packet loss directly influence the user's experience of a game or video streaming application. If the latency is high, the game or video might appear to lag. If the jitter and packet loss are high the game state could be inconsistent between devices and the video will appear inconsistent and possibly distorted.

Test Setup

PongWifiP2P measures the round-trip latency and packet loss of Pong game update packets. Jitter is calculated by differentiating the round-trip latency. The latency, jitter and packet loss are measured as follows:

1. Connect the two devices with *WifiP2PApp*.
2. The master and slave devices start sending update packets.
3. Each packet contains game state updates, a packet number, a timestamp and a CRC checksum. The received packets are checked for correctness by calculating and comparing the CRC checksums. Incorrect packets are discarded.
4. If the latest received packet number differs from the previous successfully received packet number by more than one, the packets lost counter is incremented with the difference in packet numbers. As every packet contains game state updates, only the latest packet is relevant. Packets arriving in incorrect order will be discarded and only the newest packet will be used.
5. Upon receiving an update packet, a reply packet containing the original packet timestamp is returned to the sender. Comparing the received timestamp with the current time at the original sender's side, the round-trip latency can be calculated.
6. Perform this test at 0 m, 10 m, 20 m, and 30 m device separation.
7. Repeat this procedure by using the Wi-Fi interface, similar to the throughput tests.

Each test was run until 100 000 update packets were received per device pair.

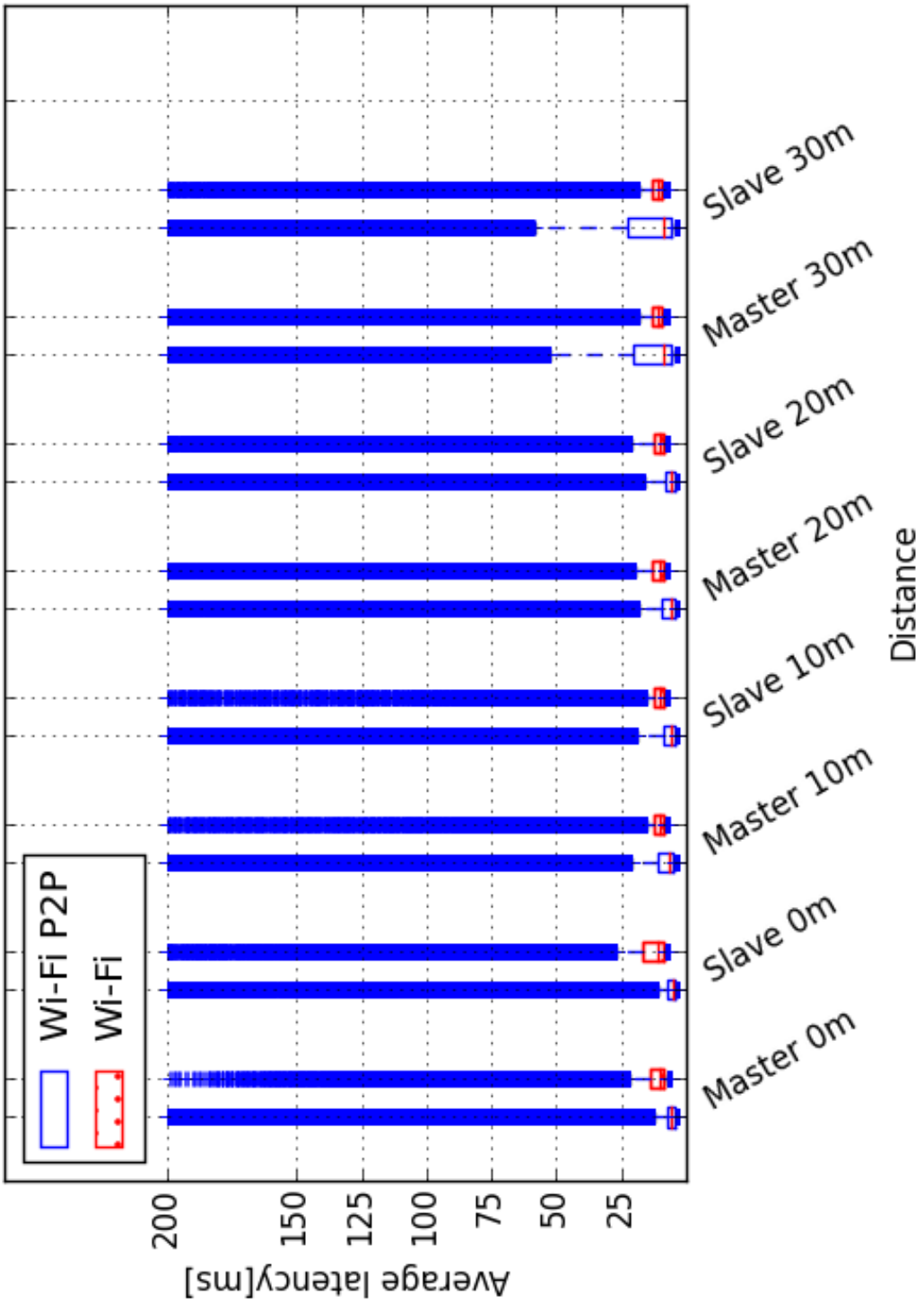

Figure 6.4: Latency comparison between Wi-Fi P2P and standard Wi-Fi with device separation distances between 0 m and 30 m. The boxes represent the 25th and 75th percentile and the red line indicates the median value. Caps indicate the 10th and 90th percentiles. Outliers outside the 10th and 90th percentiles are indicated with plus signs.

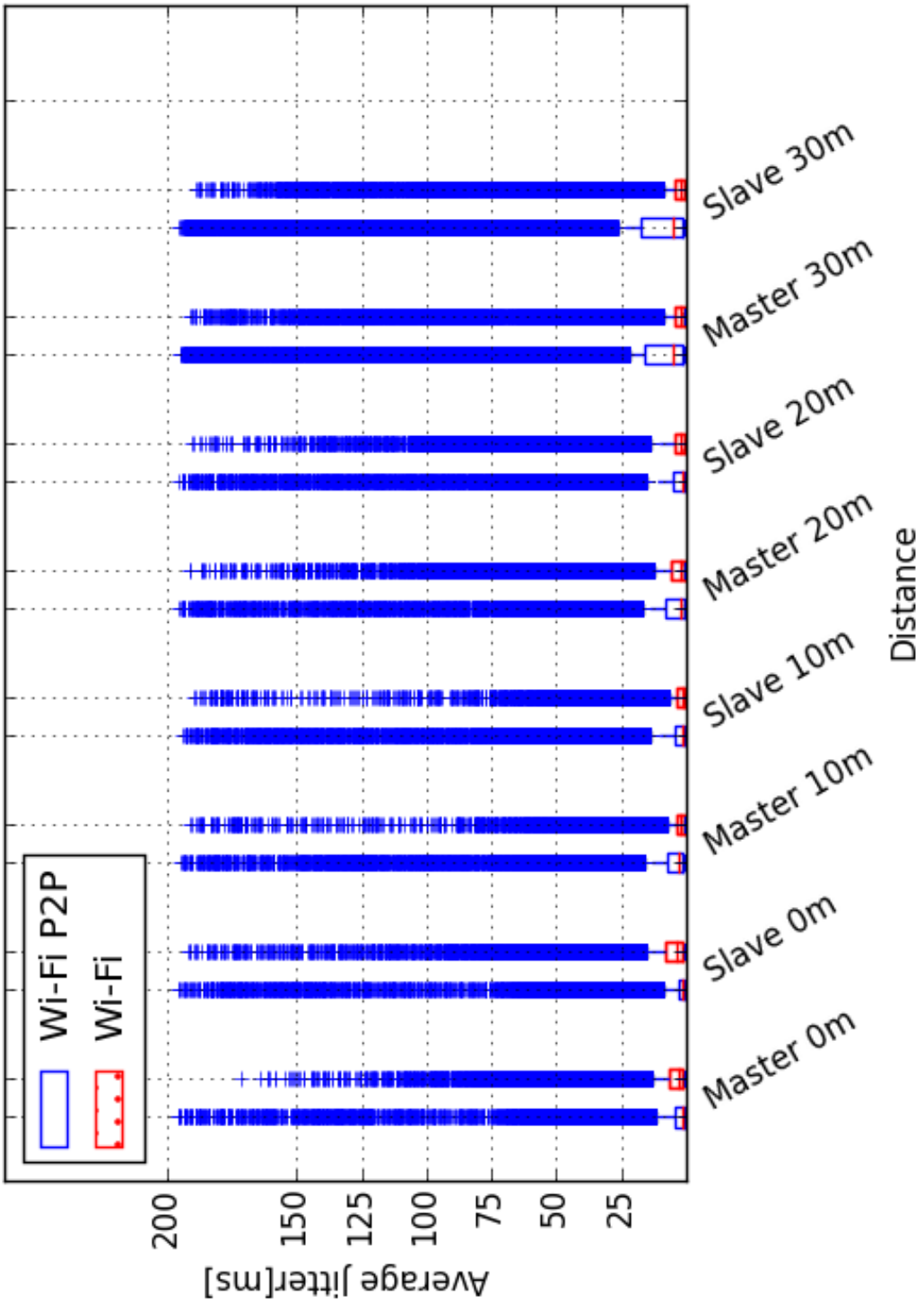

Figure 6.5: Jitter comparison between Wi-Fi P2P and standard Wi-Fi with device separation distances between 0 m and 30 m. The boxes represent the 25th and 75th percentile and the red line indicates the median value. Caps indicate the 10th and 90th percentiles. Outliers outside the 10th and 90th percentiles are indicated with plus signs.

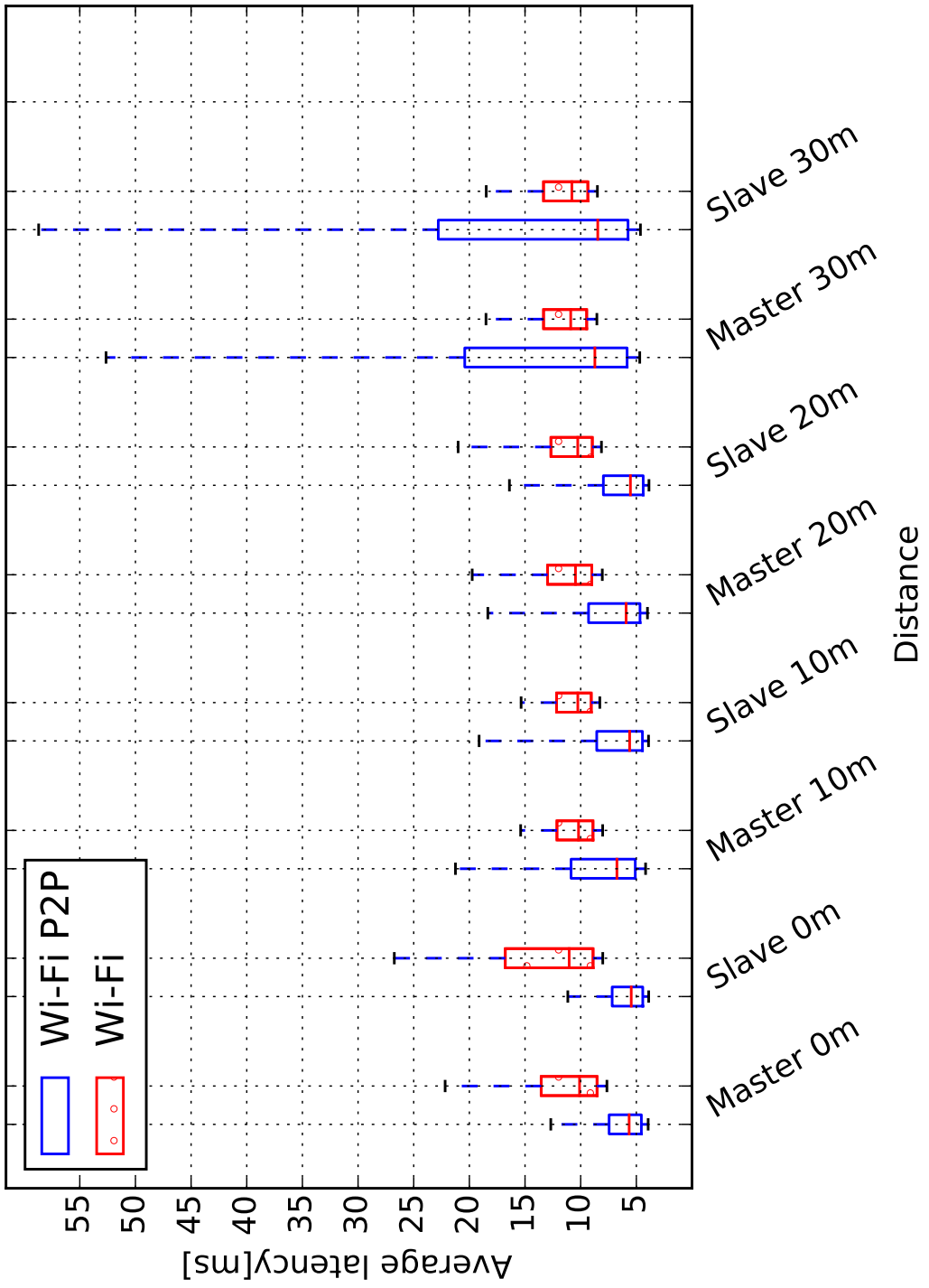

Figure 6.6: Latency comparison between Wi-Fi P2P and standard Wi-Fi with device separation distances between 0 m and 30 m. The boxes represent the 25th and 75th percentile and the red line indicates the median value. Caps indicate the 10th and 90th percentiles. Outliers are hidden.

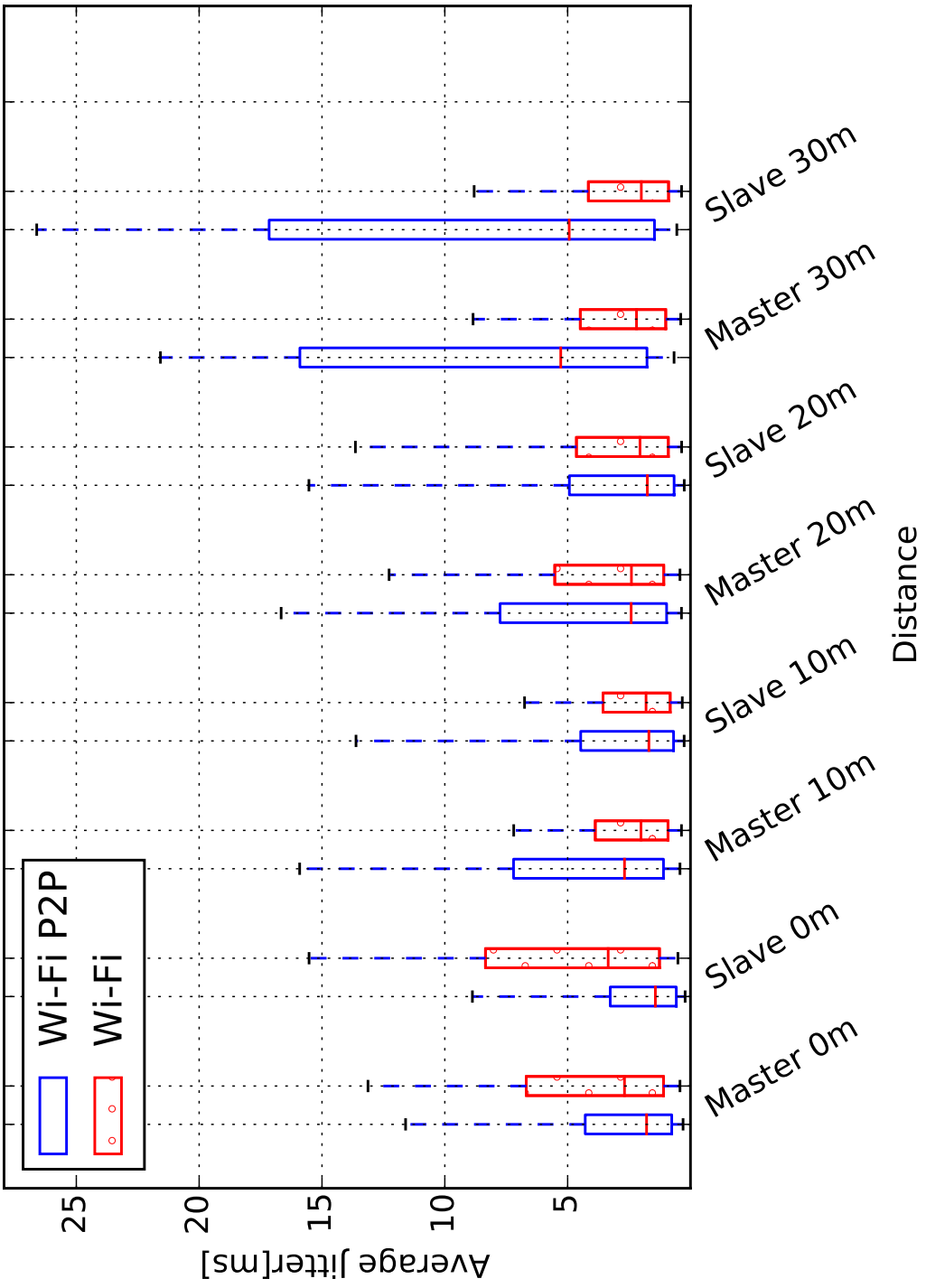

Figure 6.7: Jitter comparison between Wi-Fi P2P and standard Wi-Fi with device separation distances between 0 m and 30 m. The boxes represent the 25th and 75th percentile and the red line indicates the median value. Caps indicate the 10th and 90th percentiles. Outliers are hidden.

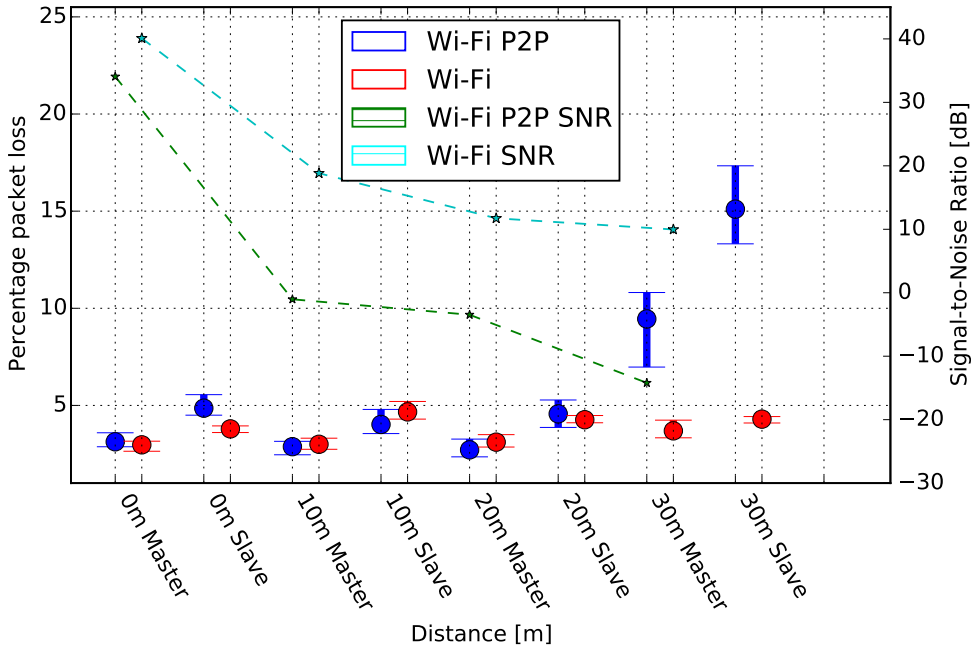

Figure 6.8: Packet loss comparison between Wi-Fi P2P and standard Wi-Fi with device separation distances between 0 m and 30 m. The error bars indicate the maximum and minimum values and the dot shows the mean.

Table 6.5: Wi-Fi P2P and Wi-Fi Signal-to-Noise Ratios in decibels (dB), averaged between the three devices. Distances for Pong tests.

	0 m	10 m	20 m	30 m
Wi-Fi P2P	34.07	-1.03	-3.47	-14.20
Wi-Fi	40.10	18.83	11.73	9.97

Expectation

Latency and jitter are expected to remain roughly constant as the increase in devices separation in terms of the speed of light is negligible, the test set-up remains the same and resends are not allowed in UDP. Similar latencies are expected for the slave and master. The packet loss is expected to increase as distance increases due to decreasing SNR, resulting in more lost packets. A finite communication distance for limited power Wi-Fi P2P is expected in the range of 40 m and standard Wi-Fi in the range of 250 m.

Results

Firstly, Figure 6.4 and Figure 6.5 show that both latency and jitter have a very wide range of values: from zero to 200 ms. However, we are more interested in the general trends of latency and jitter. For this reason, we hide the outliers in Figures 6.6,6.7.

Figure 6.6 shows the average latency remaining near constant for distances smaller than 20 m. Wi-Fi P2P latencies for devices further than 20 m increase rapidly. Standard Wi-Fi latency is slightly higher than Wi-Fi P2P, because of the two hop transmit path, i.e. from the device to the Wi-Fi AP and then to the other device, instead of directly between the devices.

Similar to latency, the average jitter (shown in Figure 6.7) remains consistent for small distances, and then for Wi-Fi P2P increases rapidly after 20m.

The 10th and 90th percentile of Wi-Fi jitter is generally lower than that of Wi-Fi P2P.

The Wi-Fi P2P packet loss percentages (shown in Figure 6.8) remain steady and then rapidly increases from 20 m onwards. Wi-Fi packet losses are always lower than Wi-Fi P2P, and increase very slowly as distance increases.

The SNR values decrease as distance increases, with Wi-Fi P2P SNR decreasing much faster than standard Wi-Fi.

The latency and jitter mean and margin of error are summarised in Tables 6.6, 6.7. The packet loss mean, minimum and maximum values are summarized in Table 6.8. Packet loss margin of error was not calculated, as the packet loss was aggregated over the entire test, instead of separate values (like latency). Packet loss values were averaged between the three device pairs and the minimum and maximum values of the three sets are given.

In all three figures, values for latency, jitter and packet loss at 0 m are higher than further distances, which is an anomalous result, as devices close to each other should perform the best because of the lower SNR. See also discussion.

Discussion

Latency and jitter performed roughly as expected, except for the sharp latency increase and jitter increase in Wi-Fi P2P for larger distances. This counter-intuitive result can be explained by multipath phenomena, i.e. because of multiple copies (reflections off walls etc.) of the packets arrive at the receiving end. Because packet loss is high for large distances, some of the faster packets get lost in transmission, and therefore allow the slower travelling reflections to be accepted as valid packets. Multipath transmissions are usually unwanted in communication systems, and more thorough testing should be conducted in open spaces to eliminate multipath errors.

Table 6.6: Pong Wi-Fi P2P and Wi-Fi latency mean and margin of error.

Distance	Mean [ms]	Margin of Error [ms]
0 m Wi-Fi P2P Master	8.61	0.08
0 m Wi-Fi Master	15.82	0.09
0 m Wi-Fi P2P Slave	8.48	0.09
0 m Wi-Fi Slave	13.79	0.06
10 m Wi-Fi P2P Master	11.33	0.09
10 m Wi-Fi Master	11.67	0.04
10 m Wi-Fi P2P Slave	10.30	0.11
10 m Wi-Fi Slave	11.88	0.05
20 m Wi-Fi P2P Master	9.93	0.09
20 m Wi-Fi Master	14.25	0.09
20 m Wi-Fi P2P Slave	9.88	0.12
20 m Wi-Fi Slave	14.64	0.11
30 m Wi-Fi P2P Master	20.49	0.17
30 m Wi-Fi Master	14.56	0.09
30 m Wi-Fi P2P Slave	21.76	0.20
30 m Wi-Fi Slave	14.90	0.11

Table 6.7: Pong Wi-Fi P2P and Wi-Fi jitter mean and margin of error.

Distance	Mean [ms]	Margin of Error [ms]
0m Wi-Fi P2P Master	4.29	0.04
0m Wi-Fi Master	5.13	0.04
0m Wi-Fi P2P Slave	3.90	0.06
0m Wi-Fi Slave	6.20	0.05
10m Wi-Fi P2P Master	5.72	0.05
10m Wi-Fi Master	3.25	0.03
10m Wi-Fi P2P Slave	4.79	0.07
10m Wi-Fi Slave	3.16	0.03
20m Wi-Fi P2P Master	5.80	0.05
20m Wi-Fi Master	4.83	0.04
20m Wi-Fi P2P Slave	5.24	0.07
20m Wi-Fi Slave	4.79	0.06
30m Wi-Fi P2P Master	10.88	0.10
30m Wi-Fi Master	4.17	0.05
30m Wi-Fi P2P Slave	12.00	0.12
30m Wi-Fi Slave	4.30	0.06

Table 6.8: Pong Wi-Fi P2P and Wi-Fi packet loss percentages.

Distance	Mean [%]	Min [%]	Max [%]
0 m Wi-Fi P2P Master	3.13	2.87	3.59
0 m Wi-Fi Master	2.97	2.64	3.16
0 m Wi-Fi P2P Slave	4.86	4.50	5.55
0 m Wi-Fi Slave	3.79	3.60	3.95
10 m Wi-Fi Master	3.00	2.74	3.31
10 m Wi-Fi P2P Master	2.88	2.46	3.15
10 m Wi-Fi P2P Slave	4.02	3.55	4.79
10 m Wi-Fi Slave	4.67	4.30	5.20
20 m Wi-Fi P2P Master	2.71	2.35	3.27
20 m Wi-Fi Master	3.11	2.86	3.50
20 m Wi-Fi P2P Slave	4.57	3.87	5.28
20 m Wi-Fi Slave	4.26	4.11	4.48
30 m Wi-Fi P2P Master	9.45	6.97	10.81
30 m Wi-Fi Master	3.69	3.33	4.24
30 m Wi-Fi P2P Slave	15.10	13.32	17.33
30 m Wi-Fi Slave	4.29	4.10	4.43

Wi-Fi jitter has a smaller spread than Wi-Fi P2P jitter, indicating a more stable and consistent connection.

Wi-Fi and Wi-Fi P2P packet loss remains low, below 5% for smaller distances. However at 30 m, Wi-Fi P2P shows a large packet loss percentage, while standard Wi-Fi packet loss only increased slightly. This indicates the drop-off point for communication distance over Wi-Fi P2P. Further than 30 m distance, the Wi-Fi P2P connection disconnected.

An anomaly occurs at 0 m in the latency, jitter and packet loss results: according to the SNR values, the best performance is expected, but we observe worse performance than at 10 m. This result can be explained by the near-field interactions of the Wi-Fi antennae. According to [35] the near-field of 2.4 GHz antennae are up to two wavelengths from the antenna, which is 25 cm. As the devices were placed next to each other for the 0 m test, near-field interactions inevitably occurred. When two antennae are in each others near-field, transmission is done via inductive coupling instead of pure electromagnetic waves. Protocols designed for larger distances could possibly not expect such interactions and therefore function ineffectively.

Communication distance over Wi-Fi P2P was much shorter than expected: according to Wi-Fi Alliance [39], the range of Wi-Fi P2P is close to 250 m, but according to this test, the maximum communication range is limited to 30 m. This could be a limitation on mobile devices, but is definitely a consideration when implementing applications which use Wi-Fi P2P.

6.5 Testing of Chat App

WifiP2P_Chat was used to demonstrate that the framework can connect multiple devices and that the generic TCP channel can distribute messages across multiple devices. It was found that only certain devices (only the Smart4 in our case) can act as Wi-Fi P2P GO with multiple Wi-Fi P2P clients. This can probably be explained by the newer hardware installed on the Smart4, which was released in 2014, instead of 2012 as the other two devices. For this reason the Smart4 was used as the Wi-Fi P2P GO in all of the chat tests.

Test Setup

The *WifiP2P_Chat* application measures accuracy of Wi-Fi P2P chats in terms of deletions and transpositions. The chat accuracy was obtained by following these steps:

1. Connect three devices (Google Nexus, Samsung Mini and Smart4) using *WifiP2PApp*. The Smart4 was the Wi-Fi P2P GO as it is the only device that could support multiple Wi-Fi P2P Clients.

2. Start the *WifiP2P_Chat* application on all three devices.

3. Each device then "speaks" at Gaussian random intervals centred around five seconds and limited to between one and ten seconds. The chat string was built from Oxford Dictionary's 100 most used words [32], uniformly randomly selected with message lengths between one and six words. These parameters were chosen to initiate a pseudo-conversation between the connected devices.

4. Each device delivers 10 messages, amounting to 30 messages per test. All chat messages are displayed in the chat window as soon as it is received. The chats are logged and compared after the test has been completed.

After the tests are completed the chat logs are compared in pairs using Python. Messages not contained in both logs are flagged as deletions and messages out of sequence between logs are flagged as transpositions. Transposition distance is also calculated.

Expectation

It is expected that most messages will be received on all three devices as the communications channel used is a connection-orientated TCP channel. It is also expected that the messages should appear in the chat log in the same order on all of the devices.

Table 6.9: Deletion and transposition count of P2P Chat between three devices. The deletion and transposition count is expressed as a percentages of the total number of messages sent. The average distance of transposed messages is given with the margin of error associated with the transposition distance.

Devices	Deletion %	Transposition %	Avg Distance	ME
Smart4 : Nexus7	0.00%	1.4%	1.18	0.104
Smart4 : Samsung Mini	0.01%	1.2%	1.23	0.134
Nexus7 : Samsung Mini	0.01%	2.2%	1.14	0.076

Results

Table 6.9 show the percentage deleted and transposed messages. The results show that few messages were deleted and that the most deletions and transpositions were between the Samsung Mini and the Nexus 7.

In general, deletions are low and most transposition distances of messages close to one.

The tests were run 100 times, with each device sending 10 messages per chat. This amounts to 3000 messages sent in total. The margins of error for the transposition distance are also given in Table 6.9.

Discussion

As was expected, the deletion percentage of the chat was very low (almost 0%). Transposition percentages are also low (between one and two percent), which means that most of the chat is correctly sequenced.

The small number of deletions (instead of perfect transmission as TCP guarantees) occurred when the Wi-Fi P2P connection was dropped and re-established, resulting in a temporary disrupt of communications. Further improvements to the framework can include the buffering of messages to negate this problem.

The deletion and transposition percentages were the highest between the Samsung Mini and Nexus 7, which could be explained by the Samsung Mini and Nexus 7 being the outer most devices: i.e. the two devices not directly connected, but rather connected via the P2P GO Smart4. This could be a result that application developers would like to consider when designing applications supporting multiple devices.

Finally, it can be seen that the average transposition distance is between 1 and 2, which is generally close enough to the original message location so that the user would not experience this transposition as a problem.

6.6 Testing of Power Usage

The power use of the framework was tested to obtain a estimate for energy use and its effect on battery life.

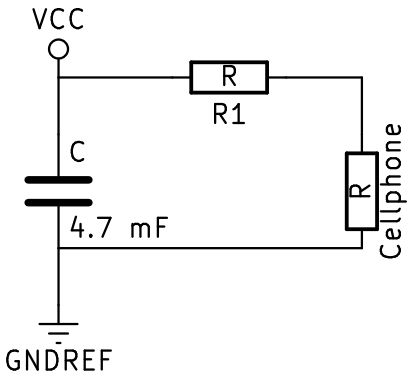

Figure 6.9: Power measurement setup. The cellphone battery was replaced by the circuit shown in the diagram. The voltage over the cellphone battery terminals and the current flowing through the cellphone was used to determine the power usage. The 4.7mF capacitor was used to stabilise the output of the power supply.

Test Setup

The test setup for measuring power usage is shown in Figure 6.9. The battery was replaced by the circuit shown in the diagram. The power usage in various states of the Wi-Fi P2P framework was measured using the current flowing in resistor R1 and the voltage measured across the cellphone. The effect of the series resistance is negligible, because the internal resistance of the cellphone is significantly larger that 1 Ohm. This is indicated by a voltage drop over the cellphone being on average 15-20 larger than over the resistor.

Tests were only performed on the on the Samsung Mini, but further testing will be needed to verify and extend the insights of the initial tests. The Nexus device has a built-in battery which could not be accessed. The Smart4 does not allow the battery to be replaced, except with a special battery clone circuit, unavailable to us.

Table 6.10: Samsung Mini power usage during different states of P2P connection.

State	Avg Power [W]	Std	% increase over Idle	% P2P over Wi-Fi
Idle (screen on)	1.105	0.091	N/A	N/A
Scanning				
Wi-Fi P2P	1.686	0.226	52.5	15.7
Wi-Fi	1.457	0.557	31.9	N/A
Bluetooth	1.275	0.174	15.4	N/A
Connected				
Wi-Fi P2P GO	2.291	0.338	107.4	61.2
Wi-Fi P2P Client	1.825	0.235	65.1	28.3
Wi-Fi	1.422	0.303	28.7	N/A
Downloading				
Wi-Fi P2P	2.788	0.609	152.3	23.4
Wi-Fi	2.259	0.424	104.4	N/A
Bluetooth	1.753	0.166	58.7	N/A

Table 6.11: Throughput to power ratio.

Medium	Power use [W]	Throughput [MB/s]	MB/s per W
Wi-Fi P2P	2.788	4.105	1.472
Wi-Fi	2.259	1.912	0.846
Bluetooth	1.753	0.160503	0.092

Expectation

It is expected that the power use will increase as the device needs to handle more processes. Power use is expected (in ascending order): scanning for P2P services should be the least energy intensive, then connected over P2P, and downloading over Wi-Fi P2P should be the most energy intensive. Furthermore, it is expected that the Wi-Fi P2P GO will consume more that the Wi-Fi P2P Client, as the GO needs to administrate the P2P network.

It is also expected that Wi-Fi P2P discover will consume more power than standard Wi-Fi scanning, as Wi-Fi P2P scanning also includes service discovery, which is not built into the standard Wi-Fi protocol.

Finally, Bluetooth is expected to consume the least amount of energy.

Results

The test results for the Samsung Mini are summarised in Table 6.10. Standard deviations are given to show the variance of the power measurements.

As was expected, the power usage increases as the cellphone CPU needs to perform more work, with the maximum power usage being when the device is downloading over Wi-Fi P2P.

The results show that Wi-Fi P2P scanning consumes more energy than standard Wi-Fi scanning. Bluetooth scanning on the Samsung Mini is very energy efficient compared to the Wi-Fi based methods.

The results from the Samsung Mini show that the Wi-Fi P2P GO consumes more energy than the Wi-Fi P2P client. and should therefore be verified by more devices.

While downloading, Bluetooth is still the most energy efficient. However, using results from the throughput tests, Table 6.11 shows the throughput to power ratio. It shows that Wi-Fi P2P has the most efficient throughput to power ratio.

It can be seen from Table 6.10, that the various states consume much more energy (15 - 150%) more than the idle state. When comparing Wi-Fi and Wi-Fi P2P, the percentage increase of Wi-Fi P2P over Wi-Fi is also given.

Discussion

The power increase in terms of percentages gives an indication as to how much battery life will be affected. For example, the Samsung Mini has a 1500 mAh battery, which on idle, would deliver roughly 7.5 hours of battery life (with the screen active). If the device is downloading over P2P for the entire time, battery-life would be reduced to 3 hours.

Compared to all the other states, Samsung Mini has a very efficient idle state and therefore the energy comparison to idle state is not necessarily appropriate, as the device will consume more energy when the user is interacting with it on a daily basis. Wi-Fi P2P is therefore also compared to standard Wi-Fi which shows that Wi-Fi P2P is more energy intensive, but only 20-30% more so than standard Wi-Fi use, except when hosting the Wi-Fi P2P group.

Bluetooth scanning and downloading are more efficient than either Wi-Fi protocol, but at the cost of throughput. The throughput to power ratio of Wi-Fi P2P is the highest, but it is important to evaluate the power use of Wi-Fi P2P and Wi-Fi during their connection procedure, especially Wi-Fi P2P which takes considerably longer to connect (discussed in the next section). For this reason, the benefits of using Wi-Fi P2P could only become profitable when files are sufficiently large.

These results provide an estimate for power usage, but it is still limited in scope and should be verified more extensively, with a wide range of devices.

6.7 Testing of Connection Time

Wi-Fi P2P connection time was tested to evaluate how quickly the P2P framework can adapt to a dynamically changing network.

Test Setup Wi-Fi P2P

The connect time and connect success rate was measured by logging the scan start time and connect time (if any) in the *WifiP2PApp*. All buttons on the UI were pressed programmatically, except for the connection acceptance dialog which is required by Android to be pressed manually. Devices were tested in pairs and these steps were followed during testing to obtain the connection time distributions:

1. Press Consumer button on one of the devices, to change it to Consumer mode. Press the Scan button to enable other devices to discover this device's advertised Generic P2P Service.

2. The other device will act as a Provider by default. Press the Discovery button on that device to start service discovery and log the time that the device started discovery. As soon as that device discovers the Consumer's Generic P2P Service it will attempt to connect.

Table 6.12: Wi-Fi P2P and Wi-Fi connection success comparison. Wi-Fi P2P once-off discovery is compared to Wi-Fi P2P continual discovery and standard Wi-Fi, in terms of connection success rate.

	Successes	Failures	Success rate	Avg Connection time [s]	ME [s]
Wi-Fi P2P	279	21	93.00%	8.92	0.289
Wi-Fi P2P Cont	291	9	97.00%	12.31	0.289
Wi-Fi	300	0	100.00%	3.75	0.172

3. Manually press the Accept button on the Consumer device when the UI dialog appears. The two devices should now be connected. Log the time at which the two devices connected.

4. Calculate the difference between discovery start time and connection established time.

Test Setup Wi-Fi

Measuring the standard Wi-Fi connection times was done by programmatically disconnecting and re-associating with a Wi-Fi hotspot.

Expectation

It was expected that Wi-Fi P2P connection times will take longer than those of standard Wi-Fi, as Wi-Fi P2P Service discovery needs to happen before the Wi-Fi P2P connection is established. A success rate of 100% is expected as the devices are connected in close proximity and connection procedure is completed identically on every attempt.

Results

Tests were performed 100 times per pair, giving 300 connection attempts in total. Figure 6.10 shows that all of the connection times, with Wi-Fi P2P continual discovery taking as long as 124 s. However, most of the connection times were below 20 s, and therefore Figure 6.11 shows this area. Standard Wi-Fi connects in 3.75 s compared to 8.92 s of Wi-Fi P2P. The connection success rate of Wi-Fi P2P was 93%, with 279 connection attempts succeeding. With continuous P2P scanning enabled, the connection success rate is 97%, but the connection time is highly variable.

Discussion

The results show that Wi-Fi P2P connection times are longer than standard Wi-Fi. The Wi-Fi P2P framework uses a longer connection sequence, i.e. service advertising, service discovery and connection hand-shaking, which explains the longer connection times than standard Wi-Fi. It was observed during testing that late service advertising (i.e. Consumer service advertising after

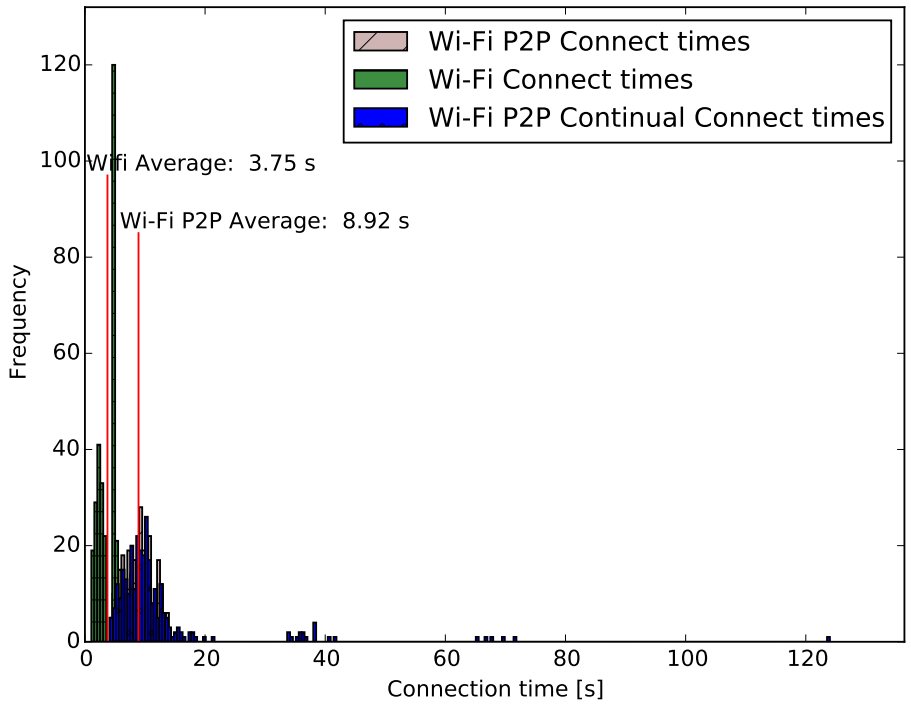

Figure 6.10: Connection time distribution with outliers show. The Wi-Fi P2P connection time distribution is hatched diagonally, the standard Wi-Fi distribution is hatched horizontally and the Wi-Fi P2P continual discovery times are hatched with dots.

the Provider has already started discovery) caused a large number of failed connection attempts. The effect of the late discovery can also be seen in the large tail-end of the Wi-Fi P2P continual discovery connection times.

Another reason for the high failure rate of P2P connections is errors within the Wi-Fi stack. During some P2P Service discovery attempts, "No service requests" error messages were received from Android's *WifiP2PManager*. According to P2Feed [21], these false messages indicating a corrupted Wi-Fi state, as a P2P service request was definitely added before discovery.

This error usually happens when the connection was initiated, the Consumer device accepted the connection, but the handshaking does not complete thus leaving the Provider device hanging in the connecting state. P2Feed suggested a work-around, but when implemented, it did not work consistently. Toggling the Wi-Fi active state resets the Wi-Fi stack and resolves this issue temporarily. This issue could not be further investigated as it is functionality provided by the Android OS. It is suspected that this bug would have to be fixed in the kernel of the operating system.

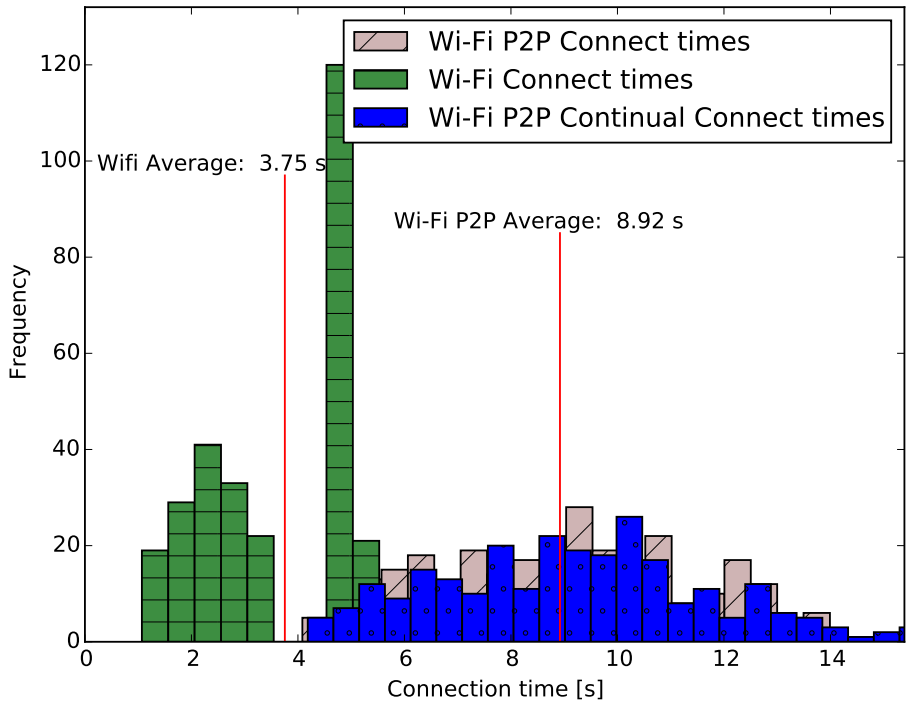

Figure 6.11: Connection time distribution with outliers hidden. The Wi-Fi P2P connection time distribution is hatched diagonally, the standard Wi-Fi distribution is hatched horizontally and the Wi-Fi P2P continual discovery times are hatched with dots.

6.8 Summary

In this chapter the framework was tested in terms of throughput, latency, packet loss, multicast support, power usage and connection time.

Throughput tests showed that downloading over Wi-Fi P2P is faster than standard Wi-Fi for distances smaller than 10 m. At 10 m, TCP communication became difficult due to the low SNR and therefore FTP tests at further distances were not conducted. Testing at larger distances and using other protocols could reveal insightful results in future work.

Latency, jitter and packet loss for Wi-Fi P2P remain very similar to standard Wi-Fi for devices separated less than 20 m. For distances of 20 m and greater, Wi-Fi P2P has significantly higher latency and packet loss than standard Wi-Fi. The Wi-Fi P2P connection was dropped at distances greater than 30 m.

The Chat tests confirmed that multiple devices can be connected to the same Wi-Fi P2P GO (if the device supports it) and can communicate with each other. Less than 3% of the messages where transposed and almost no

messages were deleted.

Power usage tests showed that Wi-Fi P2P scanning consumes more energy than standard Wi-Fi scanning and Bluetooth scanning. The Wi-Fi P2P GO consumes more energy which agrees with literature [59]. Downloading over Wi-Fi P2P consumes the most energy, resulting in a reduction in battery life of at least 60%.

It was observed that the time taken to establish a Wi-Fi P2P connection is longer than standard Wi-Fi and that Wi-Fi P2P connections are less reliable than standard Wi-Fi.

In the next chapter we will draw conclusions as to what these results mean for developing applications for Android using Wi-Fi P2P.

Chapter 7

Conclusions

In this thesis, we discussed the design and implementation of a Wi-Fi P2P framework, applications which use the P2P framework and tested the capabilities of Wi-Fi P2P by using the framework and four examples applications.

The work and contributions of this project are summarised in the next section, followed by an evaluation of the Wi-Fi P2P framework in terms of the framework requirements of Chapter 3. General suitability of Wi-Fi P2P to video streaming, downloading, social applications and games is also evaluated. Finally, we will discuss possible future work of the P2P framework, some pitfalls of Android's Wi-Fi P2P implementation and the possible future of Wi-Fi Direct in general. As before, we use Wi-Fi P2P to refer to Android's implementation and Wi-Fi Direct to refer to the protocol in general.

7.1 Summary of Work

In this section we discuss the different aspects of the project as was set out in the objectives (Section 1.4) and their contributions.

7.1.1 Wi-Fi P2P Framework

A Wi-Fi P2P framework was designed and implemented according to use-cases and specifications in Chapter 3. It was shown that the P2P framework enables applications to use Wi-Fi P2P as well as enable testing of Wi-Fi P2P capabilities. The P2P framework is modular in design so that it can be used by other applications developers to understand and implement Wi-Fi P2P applications.

7.1.2 Wi-Fi P2P Example Apps

Four example applications (*Swiftp*, *WifiP2P_FTP*, *PongWifiP2P*, and *WifiP2P-Chat*) were adapted or implemented to use the Wi-Fi P2P framework. Each

of these applications enable a use-case of Wi-Fi P2P as was mentioned Section 3.1. These applications also enabled Wi-Fi P2P capabilities to be tested, discussed in the next subsection. Furthermore, these applications give insight for applications developers using the P2P framework.

7.1.3 Wi-Fi P2P Testing

Wi-Fi P2P was tested in terms of throughput, latency, jitter, packet loss, multiple device support, energy use and connection time and connection reliability. Standard Wi-Fi and Bluetooth throughput was measured and compared to Wi-Fi P2P. Latency, jitter, packet loss and connection time were also compared to standard Wi-Fi.

WifiP2P_FTP

The *WifiP2P_FTP* application showed that Wi-Fi P2P throughput is higher for small device separation (at 0 m): 4.11 MB/s compared to standard Wi-Fi's 1.91 MB/s. This result can be explained by the devices connected directly instead of through a router. The test also showed lower Wi-Fi P2P throughput between devices that are further apart: 2.22 MB/s instead of 4.11 MB/s. This decline in data-rate can be explained by a lower signal-to-noise ratio at the receiver device and thus a greater number of packet retransmissions are needed to correctly successfully transmit the file. Wi-Fi P2P throughput was significantly higher than Bluetooth: 4.11 MB/s compared to 0.161 MB/s.

PongWifiP2P

The *PongWifiP2P* applications showed that latency, jitter and packet loss remain approximately constant (around 8-15 ms latency, 5 ms jitter and 4% packet loss) for device separation smaller than 20 m and increase rapidly for larger distances with a cut-off after 30 m. Unlike *WifiP2P_FTP* which failed at 10 m, *PongWifiP2P* was able to communicate further than 10 m, by using UDP instead of connection-orientated TCP. The reason for this effect is probably high packet loss causing large congestion of TCP packet resends on devices further than 10 m separated.

WifiP2P_Chat

Multiple device support was tested using *WifiP2P_Chat*. These tests showed that multiple devices can connect to a Wi-Fi P2P Group Owner if that device supports it. This test also showed that messages can be sent to all devices in the network using the framework's TCP channel. Messages are infrequently transposed (less than 3%) and very infrequently deleted (less than 1%).

Power Usage

Power usage tests were performed using the *WifiP2PApp* and the *WifiP2P_FTP* applications. Using results from the Samsung Mini, it showed a 52.5% increase in power use while scanning for Wi-Fi P2P services compared to an idle device. This is considerably more than standard Wi-Fi scanning which only consumes 31.9% more than idle. This difference can be explained by the more complex scanning routine of Wi-Fi P2P which include network layer P2P service discovery. The power usage increase for Wi-Fi P2P connected devices was between 65 - 107%, depending on device role. Downloading over Wi-Fi P2P using FTP resulted in most power drain (152% more than idle). It is suggested that more tests are completed, with more devices, to get a good indication of average power use.

Connection Time

Finally, the *WifiP2PApp* results showed the time to make a connection with Wi-Fi P2P to be slower (8.92 s) than that of standard Wi-Fi connections (3.75 s). This longer connection time is due to the longer P2P service discovery and longer handshaking times of Wi-Fi P2P. Wi-Fi P2P connection success rates were dependent on the sequence of the P2P scanning of the devices, i.e. Provider P2P service discovery needs to start after the Consumer advertising has completed. Better connection success rate was shown for continual Wi-Fi P2P discovery, compensating for the problem of Consumer late advertising.

7.2 Evaluation of the Wi-Fi P2P Framework

The Wi-Fi P2P framework is evaluated in this section, using the previous section's results and according to the specifications set out in Section 3.2.

High throughput

Wi-Fi P2P provides high throughput, higher even than standard Wi-Fi (4.11 MB/s compared to 1.91 MB/s). However, this bandwidth comes at a large cost to energy (up to 152% more energy than idle), and therefore Wi-Fi P2P only becomes useful if the transmitted file is sufficiently large. However, for large files such as videos or many high quality photos, Wi-Fi P2P could be a suitable choice for content sharing applications.

Low latency, jitter and packet loss

Wi-Fi P2P latency, jitter and packet loss are comparable to standard Wi-Fi, except for device separation distances of greater than 20 m. This makes Wi-Fi P2P an adequate choice for gaming applications, similar to standard Wi-Fi (when available).

Multiple device support

It was found not all devices (and hardware) support multiple connections. However, when the hardware allows, the Wi-Fi P2P framework supports multiple devices.

Dynamic network management

The Wi-Fi P2P framework provides mechanisms for continually discovering P2P services and forming connections autonomously. The framework also checks if P2P connections are alive by polling frequently.

Modular design

The P2P framework is modular in design, simplifying debugging and using framework. Furthermore, the implemented applications access the framework using the Bridge pattern, giving examples of how the framework can be used.

Reasonable energy use

Wi-Fi P2P energy use is much higher than Bluetooth and standard Wi-Fi, especially when sharing content. Careful consideration is therefore needed when downloading files over FTP to ensure that Wi-Fi P2P is a feasible option.

Except for energy use, it is concluded that the Wi-Fi P2P framework met its design specifications, albeit with some considerations such as Wi-Fi P2P's sensitivity to distance and increased energy use.

7.3 Suitability Analysis of Wi-Fi P2P for Different Applications

7.3.1 Downloading and sharing files

Wi-Fi P2P performs well for downloading and sharing files, but at a large energy cost. Only if the files are sufficiently large, would Wi-Fi P2P become a good alternative for standard Wi-Fi.

7.3.2 Video streaming

Wi-Fi P2P could perform well for video streaming, as it shows high throughput, low latency and low jitter characteristics. However, similar to downloading and sharing files, high energy use is a big consideration.

7.3.3 Social and chatting applications

Wi-Fi P2P would not be well suited to chatting applications as Wi-Fi P2P is limited in the number of users it can connect and has lower reliability than Wi-Fi. It is thus suggested that social applications, which generally do not transfer large amounts of data, use Wi-Fi or even Bluetooth as communication channel.

7.3.4 Gaming applications

Wi-Fi P2P can support gaming applications, with low latency and jitter. However, Wi-Fi P2P is very sensitive to range and very energy intensive. Bluetooth could be considered if only small amounts of data need to be exchanged.

7.4 Summary

In the previous section we qualitatively discussed the possible applications of Wi-Fi P2P. Wi-Fi P2P would be useful for file sharing and video streaming, and would be less suited to social and chatting applications. Gaming applications might also use Wi-Fi P2P as a possible communication medium.

It is important to note that despite Wi-Fi being more reliable, Wi-Fi P2P can be used even if there is no other infrastructure available.

7.5 Future Work

In this section, we discuss improvements to the framework, possible future uses of the Wi-Fi P2P framework and possible future work of this project. Possible problems with the Android Wi-Fi P2P implementation is discussed in the next section. Finally, the possible future of Wi-Fi Direct in general is discussed.

7.5.1 Wi-Fi P2P framework improvements

The main short-coming of this framework is energy use and reliability. Energy use can be limited by implementing a more efficient scanning scheduler or by using Bluetooth as a discovery helper. By using the lower power scanning of Bluetooth in combination with high throughput of Wi-Fi P2P, a more power efficient and stable framework could be obtained.

As for reliability, the Wi-Fi P2P framework follows a set connection pattern, assuming that the previous steps were successful. This could cause unexpected delays or even failed connections when the previous requirements are not met. An example of this is the case where the other device is not scanning, and this device would like to connect to it. Another example is when two devices are busy connecting, and a third tries to join the Wi-Fi P2P GO;

this does not succeed and the third device is stuck in the "Connecting" step. A state machine could be implemented to improve the connection procedure and error recovery.

Furthermore, the P2P framework does not account for a possible disconnect or temporary drop of connection. This results in the connection being disrupted at the application level, instead of only the P2P framework trying to regain connectivity. Buffering incoming and outgoing communications could improve the reliability experienced by the applications using the P2P framework, even if a disconnect occurs.

7.5.2 Future work of this project

Thorough power testing of the P2P framework and other device-to-device communication protocols would benefit this project, as the current tests do not give clear and satisfactory results. Power tests should be run on a variety of devices, and use a dedicated power measuring power supply, instead of the circuit used in Figure 6.9.

FTP tests are currently only conducted for 0 - 10 m device separation. Further distances could be tested as well as using for example UDP as the underlying transfer protocol for FTP.

Pong tests results currently display near-field interactions and multipath errors. Further testing in an open environment should be conducted to provide clearer results.

Finally, combining Wi-Fi P2P and Bluetooth technologies in a single framework could be explored.

7.5.3 Future of the Wi-Fi P2P framework

The P2P framework can be used by Android applications developers to add Wi-Fi P2P functionality to their applications, or it can be used to gain insight into using Wi-Fi P2P natively in applications. Given the results measured using the framework, applications developers will also have insight about the capabilities and limitations of Wi-Fi P2P.

7.6 Android Wi-Fi P2P pitfalls

During the development of the P2P framework on Android, we discovered functionality that could be improved and potential dangers that developers should be aware of when using the Wi-Fi P2P on Android.

Firstly, it is important to note that both devices need to be scanning in order to connect and advertise P2P Services. By default, devices scan only for a limited period of time and are therefore discoverable only for that period.

It was mentioned in Section 6.7 that the P2P Service discovery process sometimes fails due to a corrupted Wi-Fi stack. The temporary solution, i.e. resetting the Wi-Fi, is not a viable long term solution as it also interrupts normal Wi-Fi communications. This error has been logged on the Android issue tracker, however, as of Android 4.4.4 this issue has not been addressed.

Another consideration when using Wi-Fi P2P is that it uses the same Wi-Fi module as normal Wi-Fi. This means that Wi-Fi P2P traffic is multiplexed with normal Wi-Fi and could cause a drop in throughput when using both Wi-Fi and Wi-Fi P2P together. Wi-Fi P2P scanning and discovery of P2P services also use the Wi-Fi module and causes slower performance of other Wi-Fi P2P applications.

Due to limited transmitting power of Wi-Fi P2P devices, resulting in lower signal-to-noise ratio and lower throughput than normal Wi-Fi, connection range is limited.

Finally, Wi-Fi P2P is based on the 802.11 Infrastructure mode, and therefore only supports "spoke" networks, i.e. clients connect to a centralised Wi-Fi P2P. This disqualifies Wi-Fi P2P from completely peer-to-peer mesh networks.

7.7 Future of Wi-Fi Direct

Despite these challenges mentioned in the previous section, Wi-Fi Direct and its uses are meaningful and expanding.

According to Tehrani et al. [57] bandwidth use of mobile devices has increased drastically and it is expected that current 4G technologies will not scale sufficiently in the future, especially if ultra high definition video streaming become common on mobile devices. Tehrani suggests that device-to-device communications (including for example Wi-Fi Direct) could assist in distributing content and services between connected devices instead of communicating with further away 4G base stations.

According to Wi-Fi Alliance [36] many capabilities including voice quality communication (Wi-Fi Voice [38]) and context aware connectivity (Wi-Fi Aware [37]) will be included in new devices. These services all leverage Wi-Fi and Wi-Fi Direct service discovery to provide services to the user.

Although there are still various discussions happening [14], Wi-Fi Direct could also be a potential technology for connecting wearable devices and the Internet of Things.

Ultimately, Wi-Fi Direct could become the stepping stone for smartphone based opportunistic peer-to-peer mesh networks.

Bibliography

[1] "Android 4.0 Ice Cream Sandwich now official, includes revamped design, enhancements galore." [Online]. Available: http://www.engadget.com/2011/10/18/android-4-0-ice-cream-sandwich-now-official/ [Accessed: 2015-05-11]

[2] "Android crushing its competitors in South Africas mobile race." [Online]. Available: http://mybroadband.co.za/news/smartphones/127040-android-crushing-its-competitors-in-south-africas-mobile-race.html [Accessed: 2015-09-01]

[3] "Android, the world's most popular mobile platform | Android Developers." [Online]. Available: http://developer.android.com/about/index.html [Accessed: 2015-02-26]

[4] "Apple Has Shipped 1 Billion iOS Devices." [Online]. Available: http://www.businessinsider.com/apple-ships-one-billion-ios-devices-2015-1 [Accessed: 2016-01-14]

[5] "Apple store downloads 2015 | Statistic." [Online]. Available: http://www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/ [Accessed: 2016-01-14]

[6] "Bluetooth Chat Sample." [Online]. Available: https://android.googlesource.com/platform/development/+/eclair-passion-release/samples/BluetoothChat/src/com/example/android/BluetoothChat/BluetoothChatService.java [Accessed: 2015-09-21]

[7] "Confidence Interval for the Mean." [Online]. Available: http://onlinestatbook.com/2/estimation/mean.html [Accessed: 2015-09-28]

[8] "Dashboards | Android Developers." [Online]. Available: https://developer.android.com/about/dashboards/index.html#Platform [Accessed: 2015-09-01]

[9] "DatagramSocket (Java Platform SE 7)." [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html [Accessed: 2015-05-19]

[10] "Fast Facts | Bluetooth Technology Website." [Online]. Available: http://www.bluetooth.com/Pages/Fast-Facts.aspx [Accessed: 2015-02-24]

[11] "FTP Server (swiftp)." [Online]. Available: http://ppareit.github.io/swiftp/ [Accessed: 2015-05-07]

[12] "Gartner Says Smartphone Sales Surpassed One Billion Units in 2014." [Online]. Available: http://www.gartner.com/newsroom/id/2996817 [Accessed: 2015-08-08]

[13] "Global mobile statistics 2013 Part A: Mobile subscribers; handset market share; mobile operators." [Online]. Available: http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#smartphone-shipments

[14] "IDG Connect WiFi Direct v Bluetooth: Which Will Win?" [Online]. Available: http://www.idgconnect.com/abstract/9183/wifi-direct-bluetooth-which-will-win [Accessed: 2015-08-07]

[15] "IEEE SA - 802.11u-2011 Part II: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 9: Interworking with External Networks," Tech. Rep. [Online]. Available: http://standards.ieee.org/findstds/standard/802.11u-2011.html

[16] "IEEE-SA - The IEEE Standards Association - Home." [Online]. Available: http://standards.ieee.org/ [Accessed: 2015-02-17]

[17] "Jitter." [Online]. Available: https://en.wikipedia.org/wiki/Jitter#Packet_jitter_in_computer_networks [Accessed: 2016-02-04]

[18] "Learn - AllSeen Alliance." [Online]. Available: https://allseenalliance.org/developers/learn [Accessed: 2015-05-12]

[19] "Multipeer Connectivity Framework Reference." [Online]. Available: https://developer.apple.com/library/ios/documentation/MultipeerConnectivity/Reference/MultipeerConnectivityFramework/index.html#//apple_ref/doc/uid/TP40013328 [Accessed: 2016-02-07]

[20] "Our History | Bluetooth Technology Website." [Online]. Available: http://www.bluetooth.com/Pages/History-of-Bluetooth.aspx [Accessed: 2015-02-24]

[21] "P2Feed." [Online]. Available: http://p2feed.com/wifi-direct.html [Accessed: 2015-02-02]

[22] "Packet loss." [Online]. Available: https://en.wikipedia.org/wiki/Packet_loss

[23] "PeerDeviceNet - Secure sharing among your devices." [Online]. Available: http://www.peerdevicenet.net/ [Accessed: 2015-05-12]

[24] "QoS Requirements of Video Quality of Service Design Overview." [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=357102{&}seqNum=2 [Accessed: 2016-02-04]

[25] "Radiocommunication Sector (ITU-R) - ITU global standard for international mobile telecommunications." [Online]. Available: http://www.itu.int/ITU-R/index.asp?category=information{&}rlink=imt-advanced{&}lang=en [Accessed: 2014-03-12]

[26] "ServerSocket (Java Platform SE 7)." [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html [Accessed: 2015-05-19]

[27] "Smartphones: So Many Apps, So Much Time." [Online]. Available: http://www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps--so-much-time.html [Accessed: 2015-08-08]

[28] "Socket (Java Platform SE 7)." [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html [Accessed: 2015-05-19]

[29] "South Africans spend more on mobile: report." [Online]. Available: http://businesstech.co.za/news/mobile/49343/south-africans-spend-more-on-mobile-report/ [Accessed: 2014-03-13]

[30] "t Distribution." [Online]. Available: http://onlinestatbook.com/2/estimation/t_distribution.html [Accessed: 2015-09-19]

[31] "The Independent Communications Authority of South Africa (ICASA) hereby issues a warning on the use of ISM technology in the 2 , 4 GHz frequency band ."

[32] "The OEC: Facts about the language - Oxford Dictionaries." [Online]. Available: http://www.oxforddictionaries.com/words/the-oec-facts-about-the-language [Accessed: 2015-05-04]

[33] "The Pong for Android Open Source Project on Open Hub." [Online]. Available: https://www.openhub.net/p/android-pong [Accessed: 2015-05-07]

[34] "Throughput - Wikipedia, the free encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/Throughput [Accessed: 2015-09-08]

[35] "Whats The Difference Between EM Near Field And Far Field?" [Online]. Available: http://electronicdesign.com/energy/what-s-difference-between-em-near-field-and-far-field [Accessed: 2016-02-04]

[36] "Wi-Fi Alliance." [Online]. Available: http://www.wi-fi.org/discover-wi-fi [Accessed: 2015-08-07]

[37] "Wi-Fi Aware | Wi-Fi Alliance." [Online]. Available: http://www.wi-fi.org/discover-wi-fi/wi-fi-aware [Accessed: 2015-08-07]

[38] "Wi-Fi CERTIFIED Voice Programs | Wi-Fi Alliance." [Online]. Available: http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-voice-programs [Accessed: 2015-08-07]

[39] "Wi-Fi Direct | Wi-Fi Alliance." [Online]. Available: http://www.wi-fi.org/discover-wi-fi/wi-fi-direct [Accessed: 2014-03-17]

[40] "WiFi SNR - Android Apps on Google Play." [Online]. Available: https://play.google.com/store/apps/details?id=com.javali.wifisnr{&}hl=en [Accessed: 2016-02-06]

[41] "Latency," 2015. [Online]. Available: https://en.wikipedia.org/wiki/Latency_(engineering)#Packet-switched_networks [Accessed: 2015-09-08]

[42] A. Beznosyk, P. Quax, K. Coninx, and W. Lamotte, "Influence of network delay and jitter on cooperation in multiplayer games," *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, pp. 351–354, 2011. [Online]. Available: http://doi.acm.org/10.1145/2087756.2087812

[43] E. Burnette, "Patrick Brady dissects Android," 2008. [Online]. Available: http://www.zdnet.com/article/patrick-brady-dissects-android/

[44] M. Butler, "Android: Changing the Mobile Landscape," *IEEE Pervasive Computing*, vol. 10, no. 1, pp. 4–7, jan 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5676144

[45] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device to device communications with wifi direct: overview and experimentation," *IEEE Wireless Commun.*, no. June, pp. 96–104, 2013. [Online]. Available: http://enjambre.it.uc3m.es/{~}agsaaved/papers/2012_camps_wircommag.pdf

[46] M. Chetty, S. Sundaresan, S. Muckaden, and N. Feamster, "Investigating Broadband Performance in South Africa," Tech. Rep., 2013. [Online]. Available: http://www.researchictafrica.net/docs/RIA_policy_paper_measuring_broadband_performance_South_Africa.pdf

[47] B. P. Crow, I. Widjaja, J. G. Kim, and P. Sakai, "IEEE 802.11 Wireless Local Area Networks," *Communications Magazine, IEEE*, vol. 35, no. September, pp. 116–126, 1997.

[48] Y. Dalal, C. Sunshine, and V. Cerf, "Specification of Internet Transmission Control Program." [Online]. Available: http://tools.ietf.org/html/rfc675

[49] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[50] B. S. I. Group, "Bluetooth Core Version 3.0 + HS," Tech. Rep. April, 2009.

[51] J. Haartsen, "The Bluetooth Radio System," vol. 2, no. February, pp. 28–36, 2000.

[52] T. K. Paul and T. Ogunfunmi, "Wireless LAN comes of age: Understanding the IEEE 802.11n amendment," *IEEE Circuits and Systems Magazine*, vol. 8, pp. 28–54, 2008.

[53] J. Postel, "DoD standard Internet Protocol." [Online]. Available: http://tools.ietf.org/html/rfc760

[54] ——, "User Datagram Protocol." [Online]. Available: http://tools.ietf.org/html/rfc768

[55] ——, "File Transfer Protocol specification RFC 765," 1980. [Online]. Available: https://tools.ietf.org/html/rfc765

[56] J. Reynolds, "RFC 959 - File Transfer Protocol," vol. 765, no. Ien 149, pp. 1–69, 1985.

[57] M. Tehrani, M. Uysal, and H. Yanikomeroglu, "Device-to-device communication in 5G cellular networks: Challenges, solutions, and future directions," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 86–92, 2014. [Online]. Available: http://sce.carleton.ca/faculty/yanikomeroglu/Pub/ComMag-May2014-mntmuhy.pdf

[58] The Nielson Company, "THE TOTAL AUDIENCE REPORT Q1 2015," Tech. Rep., 2015. [Online]. Available: http://www.nielsen.com/content/dam/corporate/us/en/reports-downloads/2015-reports/total-audience-report-q1-2015.pdf

[59] S. Trifunovic, A. Picu, T. Hossman, and K. A. Hummel, "Slicing the battery pie: fair and efficient energy usage in device-to-device communication via role switching," in *Proceedings of the 8th ACM MobiCom workshop on Challenged networks - CHANTS '13 (2013)*, 2013, pp. 31–36. [Online]. Available: http://dl.acm.org/citation.cfm?id=2505496http://people.ee.ethz.ch/{~}apicu/files/papers/2013_tphh_chants.pdf

[60] C. van Berkel, "Multi-core for mobile phones," *2009 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1260–1265, apr 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5090858

[61] Wi-Fi Alliance, "Wi-Fi CERTIFIED Miracast." [Online]. Available: http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-miracast [Accessed: 2015-02-17]

[62] ——, "Wi-Fi Protected Setup Specification v1.0h," Tech. Rep., 2006.

[63] P. T. G. Wi-Fi Alliance, "Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.0."

Appendices

App Screenshots

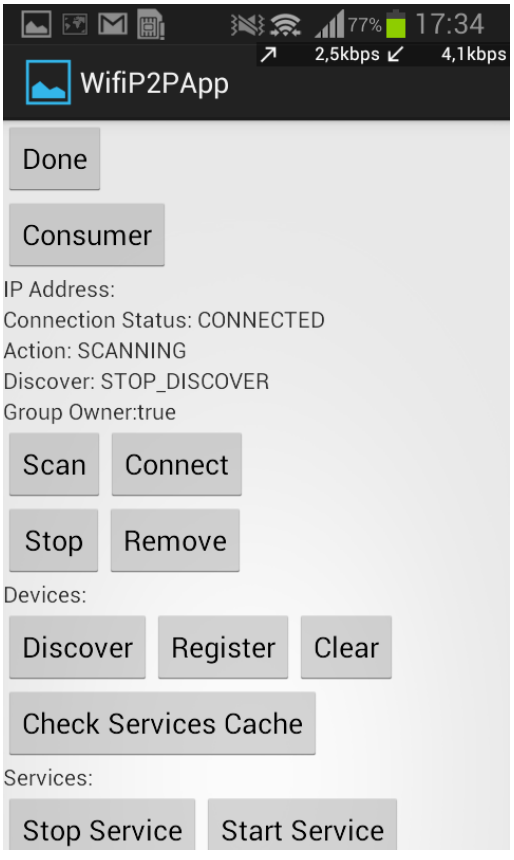

Figure 1: WifiP2PApp screenshot.

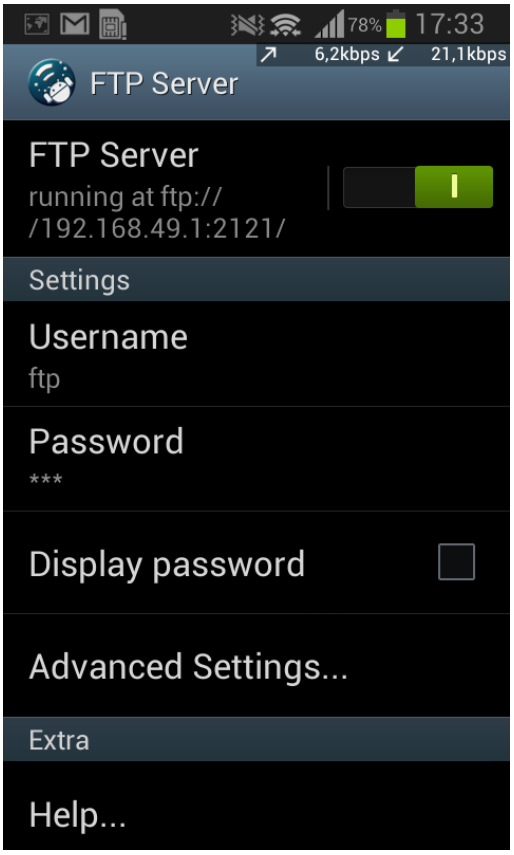

Figure 2: Swiftp screenshot.

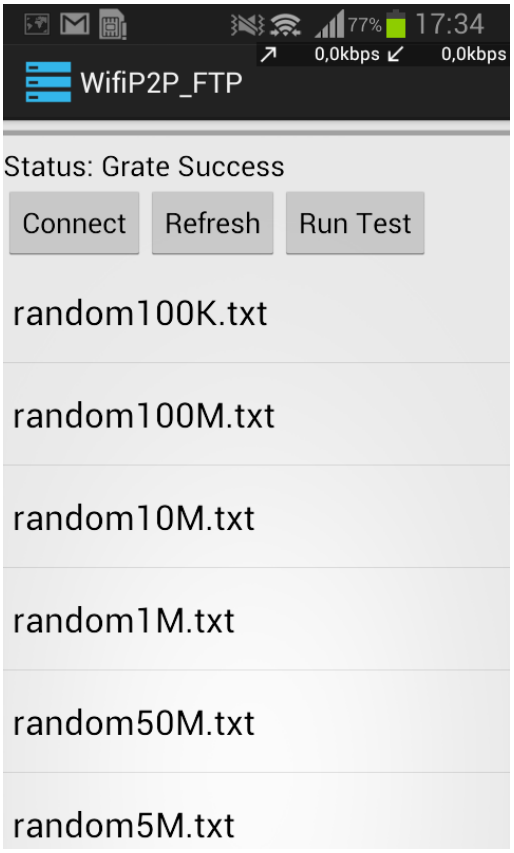

Figure 3: WifiP2P_FTP screenshot.

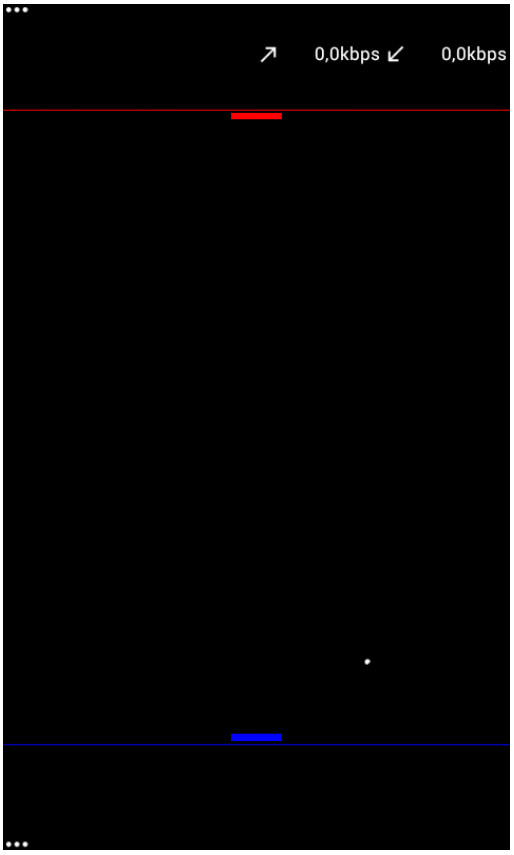

Figure 4: PongWifiP2P screenshot.

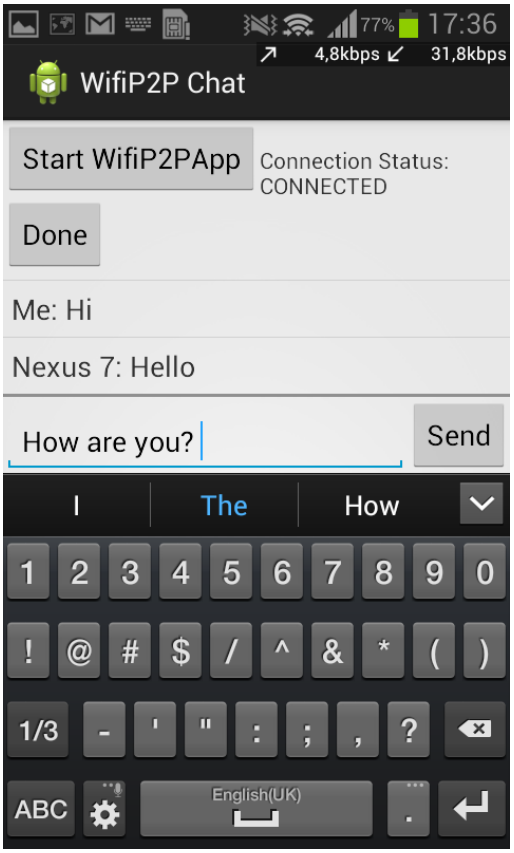

Figure 5: WifiP2P_Chat screenshot.